
GRAPHISMES

sur

IBM PC

/XT/compatibles

2D, 3D, ANIMATION, GESTION

Gabriel Cuellar

TRADUIT DE L'AMÉRICAIN
par **Bernard TORRENT**

TROISIEME EDITION
nouveau tirage


EYROLLES

61, boulevard Saint-Germain — 75005 PARIS
1987

Document de couverture

Image tridimensionnelle CADAM
(Marque déposée).

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service « Presse », Éditions EYROLLES
61, Boulevard Saint-Germain,
75240 PARIS CEDEX 05,

en précisant les domaines qui vous intéressent.
Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

Traduction autorisée de l'ouvrage américain :
« Graphics made easy . For the IBMPC and XT »
© 1984 by Gabriel CUELLAR.

Introduction

Depuis les années cinquante, beaucoup ont compris l'intérêt de l'ordinateur comme outil de création et traitement d'images. Avec sa formidable puissance de calcul, il est l'instrument idéal pour tracer des lignes droites, des rectangles et des cercles, toutes les formes géométriques standard et leurs combinaisons...

Les premiers graphiques ont été réalisés sur des moniteurs et tables traçantes en noir et blanc, et on leur trouva très vite une foule d'applications : des surfaces mathématiques peuvent être visualisées en trois dimensions, des formes complexes dessinées sous différents angles.

Les techniques cinématographiques permettent d'enchaîner et d'animer ces images, produisant des effets stupéfiants. Un autre point fort de l'ordinateur pour la réalisation d'effets spéciaux est sa capacité à générer des graphiques aléatoires.

L'apparition des écrans couleur a fait progresser les applications de façon exponentielle : il est devenu possible de traiter des images réelles pour les améliorer ou en corriger certains aspects spécifiques, ou analyser des photos prises par satellites qui seraient illisibles autrement.

Dans l'industrie, l'ordinateur rend possible l'étude d'objets — en mouvement et en couleurs — sans qu'il soit nécessaire d'en réaliser de nombreuses et coûteuses maquettes préliminaires.

Le matériel nécessaire à la conception graphique sur ordinateur est devenu abordable au particulier depuis l'apparition récente de micro-ordinateurs performants : des graphiques relativement corrects (en comparaison avec les systèmes industriels de CAO) peuvent être réalisés avec un matériel coûtant quelques dizaines de milliers de francs.

La première application individuelle a été assez naturellement l'animation des jeux vidéo.

Avec l'énorme quantité d'information circulant aujourd'hui, et parce que nous pouvons analyser des graphiques beaucoup plus rapidement que des tableaux ou du texte, l'étape logique suivante a été de présenter ces données de façon immédiatement accessible.

L'IBM-PC fournit une solution non seulement matérielle, avec sa capacité de produire des images en couleur et haute-résolution, mais aussi logicielle, avec son BASIC avancé : ce BASIC possède en effet des commandes directes pour tracer des points et des lignes, des cercles et des ellipses, des « boîtes » et des rectangles... Toutes figures géométriques qu'il est d'autre part possible de remplir avec des couleurs choisies.

Il est également possible de recopier dans des tableaux des zones définies de l'écran pour les restituer dans une nouvelle zone, rendant ainsi quasi-instantané le déplacement ou la reproduction des figures.

Nous menons dans cet ouvrage une étude approfondie des instructions graphiques (et ce qui s'y rapporte) des trois niveaux de la version 1.1 du BASIC fourni par IBM pour fonctionner sous PC DOS (Disk Operating System = Système d'Exploitation sur Disque). Bien que les instructions soient décrites en détail, il est toutefois nécessaire d'avoir quelque expérience de la programmation, de préférence dans un dialecte BASIC.

Au chapitre 1, nous décrivons les concepts de base, comme l'organisation de l'écran graphique, les pixels, les couleurs de fond et de premier plan, et le système de coordonnées. A l'aide de ces concepts simples, nous dessinons des figures, des fonctions et des lignes.

Le chapitre 2 décrit l'instruction d'affichage des lignes droites, des rectangles et des boîtes (qui sont des rectangles colorés). Nous expliquons l'utilisation de cette instruction pour tracer des fonctions avec une ligne continue, un concept qui trouvera grand nombre d'applications tout au long de ce livre. Nous parlons aussi d'une méthode de représentation d'une forme à l'aide de vecteurs, qui facilite considérablement la manipulation de figures.

Le chapitre 3 traite des formes circulaires, c'est-à-dire des cercles, des ellipses, des axes et des « camemberts ». Nous introduisons les coordonnées polaires, qui permettent de décrire de nombreuses courbes.

Au chapitre 4 on parle du coloriage de zones choisies de l'écran, de même que des différentes méthodes pour mélanger les couleurs disponibles afin de créer de nouvelles teintes.

Le chapitre 5 est une description du mode haute résolution, dans lequel on ne peut réaliser que des images noir et blanc. On signale les instructions qui se comportent différemment par rapport au mode moyenne résolution, et on donne plusieurs exemples de graphiques haute-résolution.

Au chapitre 6 nous étudions l'instruction DRAW, laquelle donne accès à tout un sous-langage spécialement orienté graphique.

Le chapitre 7 traite de l'insertion de texte sur l'écran graphique : nous expliquons comment changer la couleur des caractères, comment les placer n'importe où sur l'écran, comment les agrandir et les faire tourner, et comment réaliser un curseur clignotant, de même qu'un grand nombre de façons de donner à ces caractères des orientations inhabituelles.

Le chapitre 8 parle des transformations, c'est-à-dire des méthodes pour déplacer les graphiques en différents endroits sur l'écran, les faire tourner, les incliner et même les déformer.

Au chapitre 9 nous étudions les instructions GET et PUT, utilisées pour recopier des zones choisies de l'écran dans des tableaux de telle sorte que les images pourront être reproduites et déplacées très rapidement n'importe où sur l'écran. Ceci est très utile pour créer des fenêtres, c'est-à-dire des sections de l'écran qui sont traitées comme si elles étaient de véritables T.V miniatures, et aussi pour « traiter » l'image afin de créer des négatifs et autres effets spéciaux. Nous étudions également comment transférer le contenu de l'écran ou des parties de celui-ci sur et à partir du disque.

Le chapitre 10 est une extension du chapitre 9 à l'animation graphique. Nous commençons avec les exemples d'animation les plus rudimentaires et progressons avec des animations sur des fonds complexes, des formes changeantes, et même de l'animation sans scintillement. Nous donnons également une brève présentation de l'animation photographique, une fenêtre ouverte sur des possibilités pratiquement illimitées...

Le chapitre 11 est consacré à l'étude de graphiques en trois dimensions, contrairement aux figures plates étudiées dans les chapitres précédents. Divisé en deux parties principales, les fonctions et les figures, nous décrivons les objets tridimensionnels, représentés par des fonctions, des vecteurs ou simplement des algorithmes et nous introduisons des concepts comme les projections, les rotations, la perspective et l'élimination des lignes cachées. Nous décrivons les graphiques stéréo et, en combinaison avec les connaissances acquises au chapitre 10, nous réalisons l'animation d'objets tridimensionnels tournant dans l'espace.

Le chapitre 12 montre les histogrammes et les graphiques les plus communément utilisés pour afficher les données statistiques et comptables. Nous expliquons également les méthodes de base pour graduer les variables de telle sorte que le graphique soit toujours bien cadré, sans se soucier de la grandeur des données.

Vous trouverez enfin six appendices qui complètent le sujet : les nombres binaires, les variables Booléennes, trois programmes de hardcopy pour reproduire le contenu de l'écran sur une imprimante, les touches de fonctions du PC, les interruptions en BASIC et la table des codes ASCII.

Ce livre ne prétend pas enseigner la programmation, et on ne s'est pas étendu sur certains aspects comme la compression de code (plusieurs instructions sur une seule ligne BASIC), de même que sur l'explication par des remarques dans le corps du programme. La rareté des remarques est compensée par une explication détaillée de la méthode dans chaque cas. Les programmes ont été édités pour une compréhension facile, ainsi les lignes

```
10 FOR I=1 TO 100:IF I=S THEN IF I MOD J=0 THEN
    BEEP ELSE 20 ELSE J=J-1:S=S*2
20 NEXT
```

par exemple seront présentées dans la version plus lisible :

```
10 FOR I=1 TO 100:
  IF I=S
    THEN
      IF I MOD J=0
        THEN
          BEEP
        ELSE
          20
      ELSE
        J=J-1:S=S*2
20 NEXT
```

Remarquez que bien que certaines instructions semblent « flotter » au milieu de nulle part, elles font vraiment partie de la ligne 10 et leur position indique clairement leur relation avec le reste de la ligne. Bien que beaucoup de ces programmes peuvent trouver des applications immédiates dans nombre d'environnements, leur intention est de constituer des lignes directrices pour les programmeurs qui peuvent les façonner pour leurs besoins spécifiques. Ceci explique pourquoi on n'utilise pas de routines d'erreurs, et pourquoi les données sont rentrées de la manière la plus rudimentaire, avec l'instruction INPUT.

A la fin de chaque chapitre, nous indiquons les conditions limites des commandes introduites, de telle sorte que certaines situations inhabituelles puissent être détectées, comprises et corrigées ou évitées. Chaque chapitre contient également un certain nombre d'exercices, certains pour clarifier les concepts mentionnés brièvement, d'autres pour enrichir ces concepts.

Table des matières

Introduction	V
Table des matières.....	IX
1. Les concepts de base.....	1
Passer en mode graphique	1
L'organisation de l'écran	2
L'allumage des points	3
La détermination des coordonnées	4
La couleur	5
Le tracé de fonctions.....	8
Le tracé de lignes simples	11
2. Lignes droites et rectangles	15
Les lignes droites.....	15
Le tracé de lignes par connexion	17
Le dessin de rectangles	18
Le dessin de rectangles pleins	19
Le tracé d'une fonction avec une ligne continue	21
Méthodes primaires de tracé de droites	29
Représentation vectorielle des figures	39
3. Formes circulaires.....	49
Les cercles.....	49
Les arcs.....	54
Les ellipses	60
Les coordonnées polaires.....	63
4. La couleur.....	85
L'instruction PAINT	85
Dépassement de mémoire provoqué par PAINT	88
Cercles et ellipses pleins.....	88
Points à l'extérieur de l'écran	93
Polygones pleins	94
	IX

Camemberts pleins	94
Les couleurs non standards	96
Boîtes pleines avec la palette étendue	99
PSET avec la palette étendue	99
Le tracé de lignes avec des couleurs non standards	101
Les lignes larges	102
Extension de la palette étendue	103
Coloriage horizontal et vertical	105
Coloriage avec des couleurs non standards	107
Les couleurs avec les moniteurs noir et blanc	109
5. La haute résolution	113
Passer en haute résolution	113
L'instruction COLOR	114
PSET et PRESET	114
L'instruction LINE	114
L'instruction CIRCLE	114
La palette	115
Le tracé de fonctions	116
Les équations polaires	117
Quelques figures intéressantes	120
6. Le sous-langage DRAW	129
Lignes horizontales et verticales	129
Lignes diagonales	130
Points extérieurs à l'écran	132
Le changement de couleur	133
L'échelle	133
L'utilisation de variables comme paramètres	134
Les rotations	135
L'utilisation de chaînes comme sous-programmes	136
Le déplacement du curseur	137
Déplacement sans tracé	138
Le retour du curseur à son origine	139
DRAW compilé	140
7. Le texte dans les graphiques	143
Les curseurs	144
Position des caractères	144
Positionnement du curseur	146
Le déroulement de l'écran	147
La couleur des caractères	147
Le repositionnement des caractères	148
Mélange des caractères avec la couleur du fond	149
Caractères sans limitation de couleur	150
Caractères avec ombre	152

Les caractères larges	152
Les grosses lettres	154
Caractères n'ayant pas une orientation normale	154
Curseur logiciel	160
Sous-programme d'entrée avec un curseur clignotant	161
Impression sur la dernière position	163
Effacement jusqu'en fin de ligne	165
Effacement jusqu'à la fin de l'écran	166
L'alphabet	167
L'annotation des camemberts	174
8. Les transformations.	179
Les translations	179
L'inclinaison	181
La rotation	183
Les spirales	189
Translations non standard	192
Le changement d'échelle	194
Les distorsions.	195
9. Le transfert de parties de l'écran	201
GET et PUT	201
La copie de lignes	206
L'éditeur de lignes.	208
Le déplacement de polygones	210
La copie de cercles	211
Les fenêtres.	213
Le négatif	219
Le déroulement de l'écran.	220
Le déroulement lent	222
Bouclage et enroulement.	223
Disque et écran	228
La palette étendue avec GET et PUT.	232
La couleur aléatoire.	238
Le curseur viseur	238
La combinaison de figures.	239
Le passage d'un mode à l'autre	240
10. L'animation.	243
La balle qui rebondit	243
L'animation avec DRAW.	245
L'animation de figures complexes	246
L'animation avec un fond complexe	252
Mouvement sans scintillement avec un fond complexe	261
L'animation du fond	265
L'animation photographique.	269

11. Graphiques tridimensionnels	271
Fonctions tridimensionnelles.....	273
Objets tridimensionnels.....	308
12. Applications de gestion	397
Représentation fonctionnelle	398
L'échelle	400
L'annotation des axes	403
Graduation avec le minimum.....	406
Histogrammes simultanés	410
Barres pleines	412
Les camemberts.....	418
Appendice A. Les nombres binaires	425
Appendice B. Les variables booléennes	429
Appendice C. Vidage de l'écran sur l'imprimante	434
Appendice D. Les touches de fonctions.....	436
Appendice E. Les interruptions en BASIC.....	439
Appendice F. Les codes ASCII	441

1

Les concepts de base

1.1 PASSER EN MODE GRAPHIQUE

L'Ordinateur Personnel d'IBM (IBM PC) a deux manières différentes d'afficher des informations : le mode texte et le mode graphique. On utilise le mode texte pour afficher l'ensemble standard des caractères ASCII, plus 21 caractères supplémentaires qui permettent des graphiques limités. Le mode graphique permet un affichage composé de points et de lignes que l'on peut utiliser pour générer virtuellement n'importe quelle forme sur l'écran.

L'instruction pour passer en mode graphique¹ est SCREEN 1. Si l'ordinateur n'est pas en mode graphique quand l'instruction est exécutée, l'écran est effacé ; sinon l'instruction n'a pas d'effet notable.

Avant de commencer quelque graphique que ce soit, il est préférable d'effacer l'écran à l'aide de l'instruction CLS.

Quand on invoque BASIC, la vingt-cinquième ligne (il y a 25 lignes de texte sur l'écran) montre les cinq premiers caractères des touches de fonction ; l'instruction KEY OFF les enlève. Quand elles sont toujours présentes, n'importe quel point ou forme dessiné au-dessus les effacera ; toutefois CLS les fera apparaître de nouveau.

L'instruction KEY ON remet l'affichage des touches.

1.2 L'ORGANISATION DE L'ECRAN

L'écran graphique peut être considéré comme un tableau avec un grand nombre de lampes pouvant être allumées ou éteintes. (On les appelle en général des **pixels**, de l'anglais « **p**ictures **e**lements » (éléments d'images) on les appelle aussi des points.) Ces points sont organisés en lignes et en colonnes. Il y a 200 lignes (numérotées de 0 à 199) et 320 colonnes (de 0 à 319). Chaque point est identifié d'une manière unique par deux nombres, appelés ses coordonnées. Le premier nombre concerne la colonne et le second la ligne. La colonne la plus à gauche et la ligne la plus haute sont appelées colonne 0 et ligne 0. Le point (0, 0) est donc le point situé en haut à gauche de l'écran. Le point situé en bas à droite de l'écran aura les coordonnées (319, 199).

Le point identifié par la paire de nombres (25, 40) est situé à la colonne 25 et à la ligne 40, comme le montre la figure 1.2.

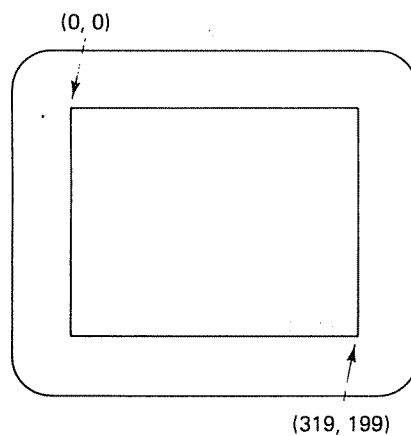


Fig. 1.1

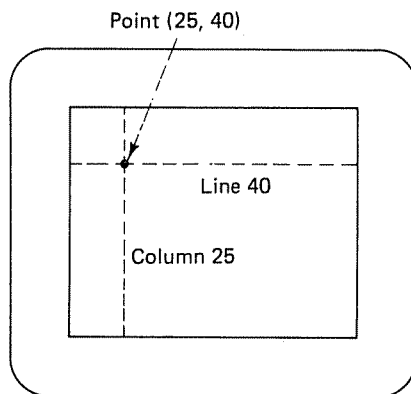


Fig. 1.2

1.3 L'ALLUMAGE DES POINTS

Quand l'écran est effacé avec l'instruction SCREEN 1 ou CLS, tous les points sont éteints. La commande pour allumer un point est PSET (*colonne, ligne*). Si *colonne* est compris entre 0 et 319 et *ligne* entre 0 et 199, le point correspondant sera allumé. (On étudiera plus tard l'effet d'autres valeurs.) Ainsi par exemple la commande PSET (50, 51) allume le point situé à la colonne 50 et à la ligne 51.

Une manière simple de créer des figures sur l'écran est de faire un dessin sur un papier quadrillé, numéroté les lignes et les colonnes et allumer les points correspondant au moyen de PSET. La figure 1.3 montre le dessin d'un visage sur un papier quadrillé et le programme 1.1 le dessine sur l'écran. A la ligne 40 du programme, la valeur 100 est ajoutée à chaque X et Y (colonne et ligne) de sorte que le graphique soit centré. La figure 1.4 montre comment le visage apparaît sur l'écran.

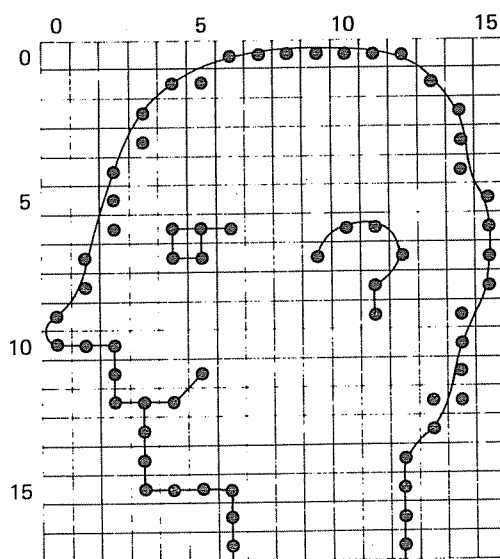


Fig. 1.3

```

0  ' ** 1-1 : Draw a face **
10 SCREEN 1:CLS
20 FOR I=1 TO 62
30   READ X,Y
40   PSET(100+X,100+Y)
50 NEXT
60 END
100 DATA 11,0,10,0,9,0,8,0,7,0,6,0,5,1,4,1,3,
        2,3,3,2,4,2,5,2,6,1,7,1,8,0,9,0,10
110 DATA 1,10,2,10,2,11,2,12,3,12,4,12,5,11,
        3,13,3,14,3,15,4,15,5,15,6,15,6,16
120 DATA 6,17,4,6,5,6,6,6,4,7,5,7

```

```

130 DATA 12,0,13,1,14,2,14,3,14,4,15,5,15,6,
          15,7,15,8,14,9,14,10,14,11,14,12
140 DATA 13,12,13,13,12,14,12,15,12,16,12,17,
          9,7,10,6,11,6,12,7,11,8,11,9

```

PROGRAMME 1.1

Bien que la création de graphiques avec cette méthode soit facile, ce n'est pas pratique pour de grands schémas à cause de l'énorme quantité de valeurs qu'il faut rentrer.

1.4 LA DETERMINATION DES COORDONNEES

Il y a deux manières de déterminer les coordonnées : en absolu et en relatif.

Quand les coordonnées sont déterminées selon le mode absolu, les valeurs *colonne* et *ligne* représentent directement la colonne et la ligne où le point doit être affiché. Si par exemple $X = 13$ et $Y = 45$ l'instruction PSET (13, 45) allumera le point situé à la colonne 13 et la ligne 45. Toutes les coordonnées qu'on a utilisées précédemment étaient données dans le mode absolu.

Quand les coordonnées sont données dans le mode relatif, les valeurs données par *colonne* et *ligne* indiquent le déplacement à partir du dernier point pris comme référence (qu'on appellera le LRP (Last Referenced Point)), c'est-à-dire le dernier point tracé avec PSET.

Pour indiquer à l'ordinateur que les coordonnées d'un point sont en mode relatif et non en mode absolu, il faut faire précéder les coordonnées par le mot STEP. L'instruction PSET STEP (10, 20) trace un point 10 colonnes à droite et 20 lignes en bas du LRP (Last Referenced Point).



Fig. 1.4

La suite d'instructions

```

10 PSET(100,90)
20 PSET STEP(15,5)

```

trace un point à la colonne 100, ligne 90 et un second point à la colonne 115 ($100 + 15$), ligne 95 ($90 + 5$).

On peut également utiliser des déplacements négatifs comme dans la suite suivante :

```
10 PSET(40,60)
20 PSET STEP(15,-10)
```

qui trace un point à la colonne 40, ligne 60 et un autre point à la colonne 55 ($40 + 15$), ligne 50 ($60 - 10$). Les valeurs positives sont ajoutées à la coordonnée correspondante, les valeurs négatives sont soustraites.

Afin d'utiliser le mode relatif pour la détermination de coordonnées, il est impératif de savoir où se trouve le LRP et comment il est affecté par chaque instruction. Voici les effets sur le LRP des instructions étudiées jusqu'ici :

PSET laisse le LRP à l'endroit exact où il trace le point.

CLS met le LRP à (160, 100) qui est considéré comme le centre de l'écran.

SCREEN 1 met le LRP à (160, 100) quand elle efface l'écran. Si l'écran est en mode graphique quand l'instruction est exécutée, cela n'a aucun effet et le LRP n'est pas changé.

1.5 LA COULEUR

Nous avons étudié comment allumer et éteindre des points, comme s'ils étaient des lampes blanches. En fait ces points peuvent être allumés ou éteints pour produire plusieurs couleurs.

Il existe deux types de couleurs² : les couleurs de fond et les couleurs de premier plan. Dans les figures que nous avons étudiées jusqu'ici la couleur du fond était toujours le noir et celle du premier plan le blanc.

1.5.1 Les couleurs du fond

On peut choisir le fond parmi 16 couleurs différentes : la couleur par défaut, c'est-à-dire la couleur du fond prise en compte par l'instruction SCREEN 1 sera le noir. L'instruction pour fixer la couleur du fond est COLOR (*bckcolor*), où *bckcolor* est un nombre compris entre 0 et 15. La table 1.1 montre la correspondance entre les codes et la couleur produite. Les nuances entre les teintes de ces couleurs dépendent de l'écran utilisé. Les couleurs de la colonne de droite de la table 1.1 sont des versions vives des couleurs de la colonne de gauche. Certains écrans video ne peuvent pas produire cette plus grande luminosité et par exemple les couleurs 1 et 9 représenteront le même bleu.

Une fois la couleur de fond fixée, l'instruction CLS vide l'écran selon cette couleur. La suite d'instructions 10 SCREEN 1 : COLOR 12 : CLS efface tout ce qu'il y a sur l'écran et le laisse coloré en rouge clair.

Au paragraphe 1.5.3 nous étudierons comment obtenir la couleur de fond avec les instructions graphiques.

1.5.2 Les couleurs du premier plan

Il y a six couleurs pour le premier plan, mais on peut n'en afficher que trois à la fois, choisies dans une des deux palettes figurant à la figure 1.2. L'instruction pour sélectionner la palette est `COLOR bckcolor, palette`. Par exemple l'instruction `COLOR 1, 0` fixera la couleur du fond en bleu (couleur du fond numéro 1) et sélectionnera la palette numéro 0 qui est verte, rouge et jaune. Si on omet *bckcolor* comme dans l'instruction `COLOR, 1`, c'est la couleur du fond précédente qui sera maintenue.

Table 1.1. — Equivalences code/couleur

<i>Code</i>	<i>Couleur</i>	<i>Code 2</i>	<i>Couleur 2</i>
0	Noix	8	Gris
1	Bleu	9	Bleu clair
2	Vert	10	Vert clair
3	Cyan	11	Cyan clair
4	Rouge	12	Rouge clair
5	Magenta	13	Magenta clair
6	Brun	14	Jaune
7	Blanc	15	Blanc vif

Table 1.2. — Les deux palettes

<i>Code</i>	<i>Palette 0</i>	<i>Palette 1</i>
1	Vert	Cyan
2	Rouge	Magenta
3	Jaune	Blanc

1.5.3 La sélection de la couleur

On peut utiliser la plupart des instructions graphiques avec d'autres couleurs que les blanc et noir que nous avons utilisés jusqu'à présent. Ces couleurs sont choisies par un code, un nombre entre 0 et 3. Dans l'instruction `PSET`, par exemple, si on ajoute un nombre après les coordonnées, ce nombre servira à fixer la couleur avec laquelle l'instruction dessinera sur l'écran. Le libellé complet de l'instruction est `PSET (colonne, ligne), couleur`. Par exemple l'instruction `PSET (100,80), 2` tracera le point (100,80) avec la couleur 2. Quand aucun code de cou-

leur n'est spécifié, c'est le code 3 qui est pris par défaut. Ainsi l'instruction PSET (1, 1) est équivalente à PSET (1, 1), 3.

Le programme 1.2 remplit l'écran avec les trois couleurs de premier plan.

```
0 *** 1-2 : Fill with colored lines **
10 SCREEN 1:CLS:C=1
20 FOR I=0 TO 319 STEP 5
30   FOR J=50 TO 100
40     PSET(I,J),C
50   NEXT:C=C+1
60   IF C=4
      THEN
        C=1
70 NEXT
```

PROGRAMME 1.2

Pour voir comment les palettes changent la couleur du premier plan, tapez l'instruction COLOR 1 ou COLOR, 0 avec les lignes créées par le programme 1.2.

Le code couleur 0 crée la couleur du fond. L'instruction PSET (100, 100), 0 efface toute couleur de premier plan au point (100, 100). Si ce point était déjà à la couleur du fond, l'instruction n'aura pas d'effet visible et ne fera que changer le LRP (dernier point pris en référence) qui deviendra le point (100, 100).

Il existe une autre instruction pour tracer des points, différente de PSET, parce que la couleur prise par défaut est celle de code 0. Le libellé de cette seconde instruction est PRESET (*colonne, ligne*). Puisqu'on peut utiliser le code couleur 0 avec PSET, les deux instructions PSET (20, 50), 0 et PRESET (20, 50) sont équivalentes.

Il n'y a pas de différence et PSET et PRESET quand on les utilise avec les couleurs 1 et 2.

Le programme 1.3 remplit lentement l'écran avec des points colorés au hasard dans des positions aléatoires.

```
0 *** 1-3 : Random dots **
10 SCREEN 1:COLOR 0,0:CLS
20 COL=1+2*RND
30 X=319*RND
40 Y=199*RND
50 PSET(X,Y),COL
60 GOTO 20
```

PROGRAMME 1.3

1.6 LE TRACE DE FONCTIONS

Une fonction est un ensemble de points qui suivent une configuration déterminée³. Nous allons utiliser l'instruction PSET pour illustrer une méthode de tracé de fonctions sur l'écran. Nous avons choisi la fonction sinus à cause de sa forme particulière. La figure 1.5 montre cette courbe entre les points π et 3π .

Dans le programme 1.4 la valeur X traverse tout l'écran (de 0 à 319). A chaque valeur de X, une valeur Y est associée, correspondant à la valeur de sinus X, et le point (X, Y) est tracé. La figure 1.6 montre le graphique produit.

```
0  "" 1-4 : Sine function ""  
10 SCREEN 1:CLS  
20 FOR X=0 TO 319  
30   Y=SIN(X)  
40   PSET(X,Y)  
50 NEXT
```

PROGRAMME 1.4

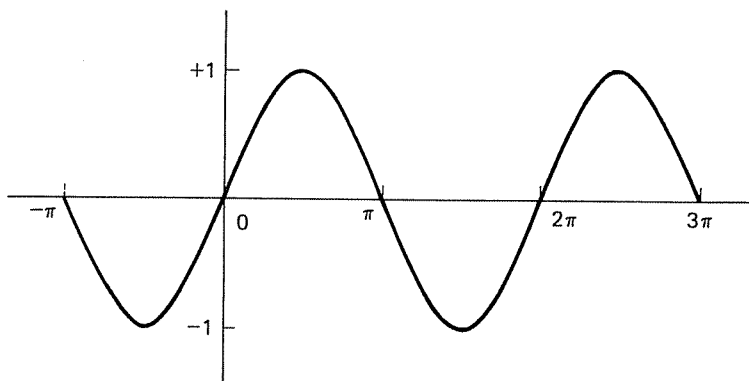


Fig. 1.5

Remarquez que la figure 1.6 n'affiche qu'une partie de la courbe sinus, parce qu'exactement la moitié de ses valeurs est négative et n'apparaît pas dans la zone de l'écran. Dans le programme 1.5 nous avons ajouté 100 à chaque valeur de Y de telle sorte que la totalité de la courbe soit tracée sur l'écran. On peut voir sur la figure 1.7 que la courbe a été déplacée vers le bas.

```
0  "" 1-5 : Sine centered ""  
10 SCREEN 1:CLS
```

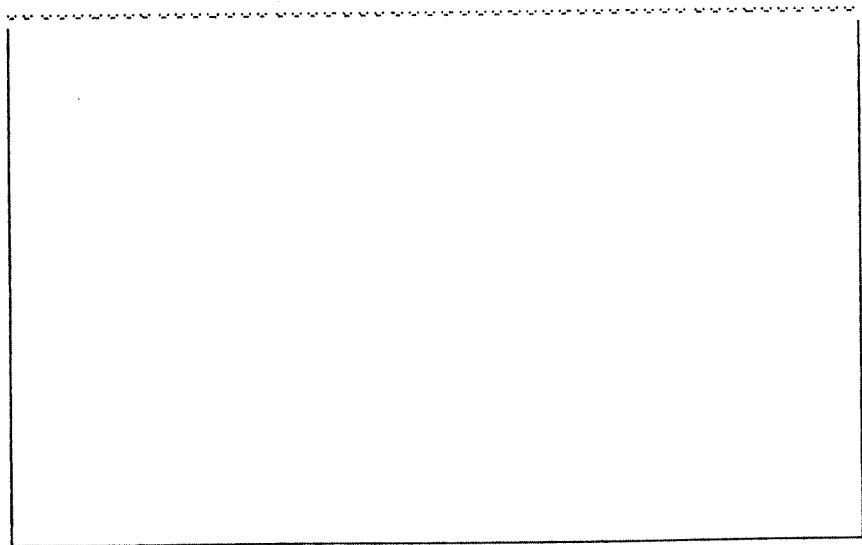


Fig. 1.6

```
20 FOR X=0 TO 319
30   Y=100+SIN(X)
40   PSET(X,Y)
50 NEXT
```

PROGRAMME 1.5

Il y a deux concepts se rapportant aux courbes (comme la courbe sinus par exemple) et qui manifestent la répétitivité :

LA FREQUENCE, qui est le nombre de courbes se trouvant dans une certaine unité. Si, par exemple, nous choisissons l'écran comme unité, nous pouvons dire que la fréquence est de 5 quand on a exactement cinq cycles complets de la courbe sinus sur l'écran.

L'AMPLITUDE, qui se rapporte à l'étendue (de haut en bas) de la courbe. Dans le cas de la courbe sinus, l'amplitude va de -1 à 1 .

La courbe tracée par le programme 1.5 a une amplitude tellement petite qu'on peut difficilement visualiser son comportement. Pour résoudre ce problème, on multiplie X par la constante $0,04$ (une valeur purement arbitraire calculée empiriquement) dans le programme 1.6, de manière à avoir deux cycles sur l'écran. Comme la courbe varie entre -1 et 1 , les valeurs $Y + 100$ induiront une fonction variant de 99 à 101 . Pour augmenter l'amplitude, on multiplie le résultat de la fonction sinus par

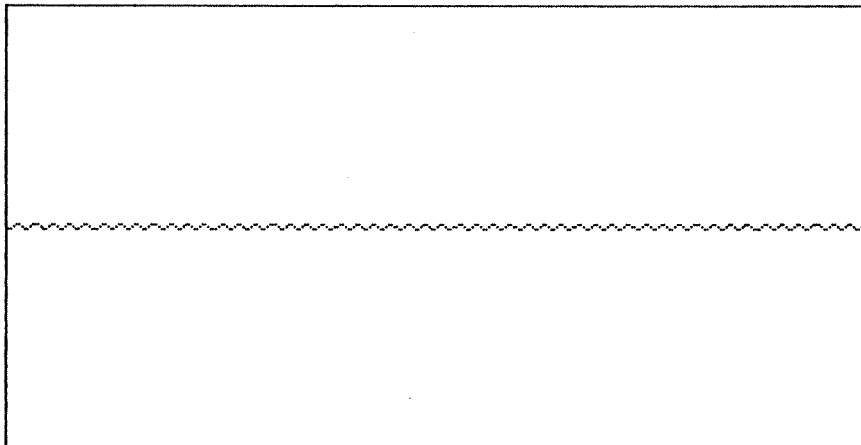


Fig. 1.7

70, ainsi le graphique varie de 30 ($100 - 70$) à 170 ($100 + 70$). La figure 1.8 montre le tracé du graphique.

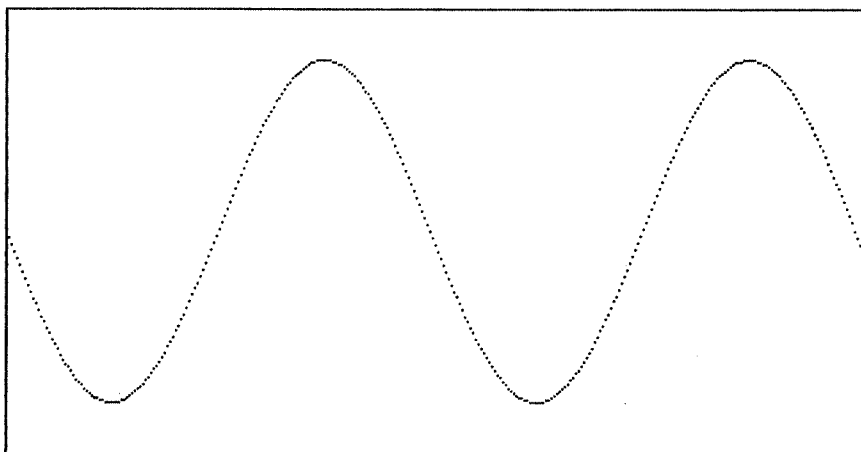


Fig. 1.8

```
0  "" 1-6 : Clear sine function ""
10 SCREEN 1:CLS
20 FOR X=0 TO 319
30   Y=100+76*SIN(X*.04)
40   PSET(X,Y)
50 NEXT
```

PROGRAMME 1.6

Au chapitre 2 nous apprendrons le moyen pour tracer cette fonction comme une ligne continue et non comme un ensemble de points disjoints, comme cela a été réalisé par le programme 1.6.

1.7 LE TRACE DE LIGNES SIMPLES

Une façon de tracer une ligne droite consiste à afficher une succession de points ; c'est ce que nous avons déjà fait dans le programme 1.2. Le programme 1.7 trace une ligne horizontale entre les points (0, 20) et 200, 20), et une ligne verticale entre les points (100, 50) et 100, 150). La figure 1.9 montre ces lignes.

```
0  '° 1-7 : Two simple lines °°  
10 SCREEN 1:CLS  
20 ' Horizontal line  
30 FOR I=0 TO 200  
40   PSET(I,20)  
50 NEXT  
60 ' Vertical line  
70 FOR I=50 TO 150  
80   PSET(100,I)  
90 NEXT
```

PROGRAMME 1.7

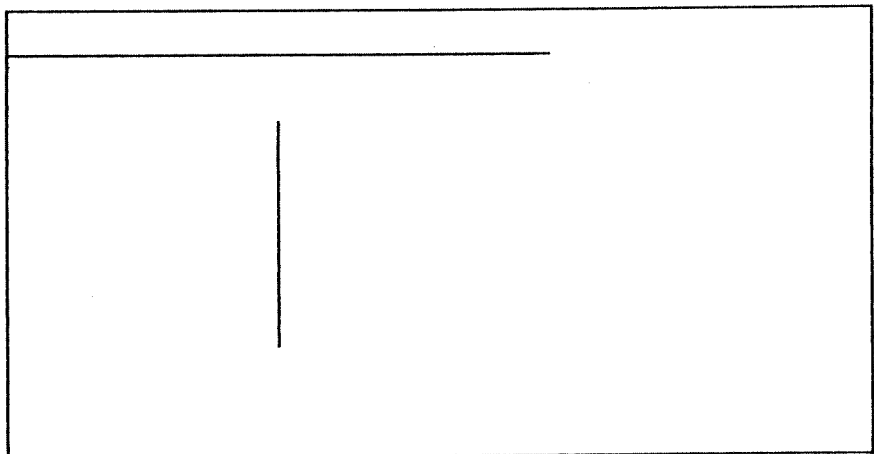


Fig. 1.9

On peut également tracer facilement des lignes diagonales avec des boucles FOR. Le programme 1.8 trace une ligne entre les points (50, 50) et (150, 150) représentée à la figure 1.10.

```
0  '° 1-8 : Simple diagonal °°
10 SCREEN 1:CLS
20 ' Diagonal line
30 FOR I=50 TO 150
40   PSET(I,I)
50 NEXT
```

PROGRAMME 1.8

Le programme 1.9 montre comment créer une diagonale plus compliquée. Il est difficile de créer des lignes qui ne soient ni horizontales, ni verticales, ni diagonales avec la même pente que celle du programme 1.8. Au chapitre 2 nous verrons comment tracer des lignes droites avec une seule instruction, et aussi bien que point par point quand on a besoin d'un contrôle total.

```
0  '° 1-9 : Not-so-simple diagonal °°
10 SCREEN 1:CLS
20 ' Complex diagonal line
30 FOR I=30 TO 180
40   PSET(40+I*0.5,I)
50 NEXT
```

PROGRAMME 1.9

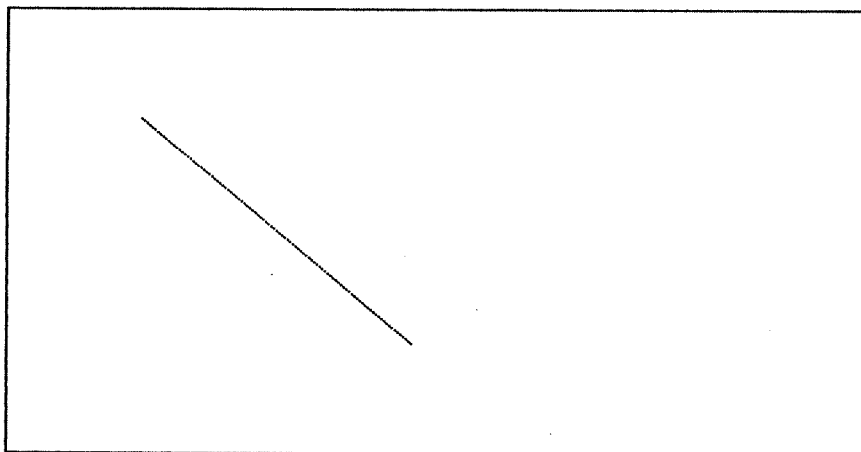


Fig. 1.10

CONDITIONS LIMITES

COLOR. Les arguments de l'instruction COLOR doivent être des nombres supérieurs ou égaux à zéro et inférieurs à 256. Les nombres avec une virgule sont arrondis, par exemple 5,4 est arrondi à 5 et 5,5 arrondi à 6. Les nombres négatifs supérieurs à - 0,5 deviennent zéro (- 0,4 devient zéro) et les nombres supérieurs à 255 et inférieur à 255,5 sont arrondis à 255.

LE FOND. Pour la couleur du fond, les nombres sont traités modulo 16 (c'est-à-dire qu'on prend le reste de leur division par 16, 16 devient 0, 17 devient 1, etc.). Si l'écran n'est pas conçu pour prendre en compte un signal d'intensité, comme nous l'avons expliqué au paragraphe 1.5.1, les couleurs de fond sont traitées modulo 8. Les nombres doivent être compris dans l'intervalle de 0 à 255 sinon on obtiendra un message d'erreur « illegal function call » (appel de fonction non autorisé).

LE PREMIER PLAN. Pour la couleur de premier plan, les nombres sont traités modulo 2, c'est-à-dire les nombres pairs produisent la palette #0 et les nombres impairs la palette #1. Le nombre doit être compris entre 0 et 255 sinon on obtiendra le message d'erreur « illegal function call ».

PSET et PRESET. Pour PSET et PRESET les arguments figurant dans *colonne* et *ligne* doivent être supérieurs ou égaux à - 32768 et inférieurs ou égaux à 32767. Les points ayant des arguments négatifs ne sont pas tracés. Les points ayant une valeur pour la ligne supérieure à 199 ne sont pas affichés. Les points (X, Y) ayant une valeur pour la colonne supérieure à 319 et inférieure à 639 sont affichés en position (X - 320, Y + 2), sauf si Y est supérieur à 197 auquel cas ils ne sont pas affichés.

Les codes couleur supérieurs à 3 se comportent aléatoirement et induisent en général la dernière couleur utilisée.

REVISION

SCREEN 1	Règle sur le mode graphique
CLS	Efface l'écran
COLOR C	Règle la couleur du fond sur la couleur C
COLOR, C	Règle le numéro de palette C
COLOR A,B	Sélectionne la couleur du fond A et la palette B.
PSET (X,Y)	Affiche le point (X,Y)
PSET (X,Y),C	Affiche le point (X,Y) avec la couleur C
PRESET (X,Y)	Affiche le point (X,Y) avec la couleur du fond.
PRESET (X,Y),C	Affiche le point (X,Y) avec la couleur C

PSET STE (X,Y) Affiche un point X unités sur la droite (sur la gauche si X est négatif) et Y unités en dessous si Y est positif) du LRP (dernier point tracé).

EXERCICES

1. Ecrire un programme pour remplir l'écran avec des visages en prenant les données du programme 1.1.
2. Ecrire un programme pour remplir l'écran avec n'importe laquelle des couleurs de premier plan.
3. Ecrire un programme qui dessine un rectangle déterminé par quatre points entrés au clavier.
4. Ecrire un programme qui dessine un rectangle déterminé par son coin supérieur gauche et son coin inférieur droit.
5. Toutes les courbes sinusoïdales tracées dans ce programme sont à l'envers (la tête en bas) ceci vient du fait que dans le système de coordonnées cartésiennes classique les valeurs de Y croissent vers le haut alors que sur l'écran elles croissent vers le bas. Changez le programme 1.6 afin que la courbe soit tracée correctement.
6. Changez le programme 1.6 afin que les parties positive et négative de la courbe soient tracées dans des couleurs différentes.

NOTES

1. Il y a en fait deux modes graphiques en BASIC, la moyenne et la haute résolution. Nous travaillerons en moyenne résolution jusqu'au chapitre 5 où nous présenterons la haute résolution.
2. Il y a un troisième type de couleur, la couleur « invisible » qui sera expliquée aux paragraphes 6.9 et 6.10.
3. C'est une définition très simpliste d'une fonction, mais suffisante pour nos besoins.

2

Lignes droites et rectangles

Au chapitre 1 nous avons étudié des méthodes élémentaires pour tracer des lignes. Ces méthodes conviennent pour des lignes très simples, mais ne sont pas utilisables pour des situations plus complexes. Dans ce chapitre nous allons étudier les différentes syntaxes des instructions pour tracer des droites, des carrés, des rectangles et des boîtes (des rectangles pleins) de même qu'un procédé rudimentaire de traçage de lignes qui peut être utile dans certains cas. Nous tracerons également quelques fonctions avec des lignes continues.

2.1 LES LIGNES DROITES

BASIC possède une instruction spécifique pour tracer des lignes droites. Sa syntaxe est `LINE (colonne 1, ligne 1) – (colonne 2, ligne 2), couleur` où *colonne 1* et *ligne 1* indiquent le point de départ et *colonne 2* et *ligne 2* le point d'arrivée ¹. *Couleur* est un code couleur pris entre 0 et 3. Si aucun code n'est précisé c'est le code 3 qui est pris par défaut. Les coordonnées de chacun des deux points peuvent être données en mode absolu ou relatif. Après l'exécution d'une instruction `LINE`, le LRP est toujours (colonne 2, ligne 2).

L'exemple suivant nous montre les combinaisons possibles de coordonnées absolues et relatives. L'instruction `LINE (0,0) – (100,100)` trace une ligne du point (0,0) au point (100,100) et laisse le LRP au point (100,100). Comme la couleur n'était pas mentionnée c'est la couleur 3

qui a été prise par défaut. Les points de départ et d'arrivée peuvent être interchangés, ainsi LINE (100,100) – (0,0) est équivalent à l'instruction précédente mais le LRP est maintenant (0,0).

La ligne PSET (50,50) : LINE (10,15) – STEP (5,10) affiche le point (50,50), et trace une ligne du point (10,15) au point (15,25) (calculé par $10 + 5$, $15 + 10$). Le LRP sera (15,25). Notez que la forme relative de (colonne 2, ligne 2) prend sa valeur par rapport au point (colonne 1, ligne 1) et non par rapport au LRP précédent.

La ligne PST (100,100) : LINE STEP (10, – 20) – (150,180) affiche le point (100,100) et trace une ligne du point (110,80) (calculé par $100 + 10$, $100 - 20$) jusqu'au point (150,80) et laisse le LRP à (150, 80).

La ligne PSET (100,100) : LINE STEP (20,30) – STEP (– 20,20) affiche le point (100,100) et trace une ligne du point (120,130) (calculé par $100 + 20$, $100 + 30$) au point (100,150) (calculé par $120 - 20$, $130 + 20$) et laisse le LRP à (100,150).

L'instruction LINE (20,10) – (120,130), 0 trace une ligne du point (20,10) au point (120, 130) avec la couleur 0, la couleur du fond. Le résultat sera une ligne qui effacera toutes formes ou points situés sur son passage.

Le programme 2.1 dessine le carré de la figure 2.1 en traçant les quatre côtés avec l'instruction LINE. Notez que la dimension des côtés horizontaux du carré (91) n'est pas égale à celle des côtés verticaux (81) ; ceci pour compenser la différence entre les dimensions verticale et horizontale de chaque pixel.

```
0  ""2-1 : Simple square ""
10 SCREEN 1:CLS
20 LINE(115,60)-(205,60)
30 LINE(205,60)-(205,140)
40 LINE(205,140)-(115,140)
50 LINE(115,140)-(115,60)
```

PROGRAMME 2.1

Le programme 2.2 remplit une partie de l'écran avec des lignes prenant leur origine au point (0,0). L'incrément initial est 1, aussi tout le carré de (0,0) à (199,199) est rempli en blanc. Au fur et à mesure que le pas croît, de moins en moins de lignes sont tracées.

```
0  ""2-2 : Fill with lines ""
10 SCREEN 1
20 FOR S=1 TO 199:CLS
30   FOR I=0 TO 199 STEP S
40     LINE(0,0)-(I,199)
```

```

50    LINE(0,0)-(199,1)
60    NEXT
70    W$=INPUT$(1)
80    NEXT

```

PROGRAMME 2.2

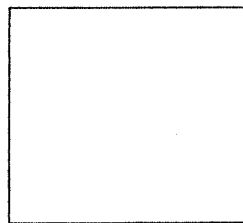


Fig. 2.1

La figure 2.2 montre le graphique produit quand S égale 5.

2.2 LE TRACE DE LIGNES PAR CONNEXION

On peut tracer une ligne à partir du LRP jusqu'à un autre point au moyen de l'instruction LINE avec la syntaxe LINE STEP (0,0) - (*colonne*, *ligne*). Comme le premier point est donné sous la forme relative et que le déplacement indiqué est zéro, l'origine de la ligne sera le LRP. Cette forme de l'instruction est si souvent employée que le point de départ de la ligne peut être omis, et si c'est le cas le point de départ sera le LRP. Le format est LINE - (*colonne*, *ligne*), *couleur*, où *colonne* et *ligne* indiquent le point qui sera connecté au LRP. Les coordonnées de ce point peuvent être sous la forme absolue ou relative.

La ligne PSET (100,80):LINE - (200,20), 3 affiche le point (100,80) et trace une ligne de ce point au point (200,20) avec la couleur 3.

Nous pouvons maintenant dessiner le carré tracé par le programme 2.1 en utilisant cette nouvelle forme de l'instruction sans avoir à répéter les coordonnées du LRP. Pour montrer deux manières d'indiquer le point final, les lignes 20 à 50 du programme 2.3 dessinent le carré en utilisant des coordonnées absolues et les lignes 70 à 100 le dessinent en utilisant des coordonnées relatives.

```

0  "" 2-3 : Square revisited ""
10 SCREEN 1:CLS
20 LINE(115,60)-(205,60)

```



```

30 LINE-(205,140)
40 LINE-(115,140)
50 LINE-(115,60)
60 W$=INPUT$(1):CLS
70 LINE-STEP(91,0)
80 LINE-STEP(0,81)
90 LINE-STEP(-91,0)
100 LINE-STEP(0,-81)

```

PROGRAMME 2.3

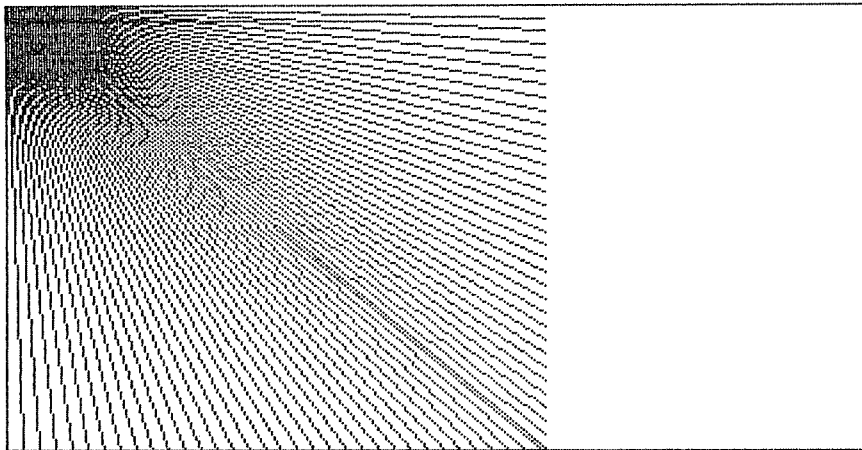


Fig. 2.2

2.3 LE DESSIN DE RECTANGLES

Outre le fait d'être capable de tracer de simples lignes, l'instruction **LINE** peut aussi dessiner des rectangles dont on précise les points des coins situés en haut et à gauche et en bas à droite. La syntaxe de cette instruction est **LINE (colonne 1, ligne 1) – (colonne 2, ligne 2), couleur, B**. Le point (colonne 1, ligne 1) indique le coin en haut à gauche du rectangle et le point (colonne 2, ligne 2) indique le point en bas à droite. Le LRP devient (colonne 2, ligne 2).

L'instruction **LINE (10,10)–(20,20),3,B** est équivalente à la suite d'instructions : **LINE (10,10):LINE–(20,20):LINE–(10,20):LINE–(10,10)**. Sous cette forme on ne peut omettre la couleur parce que l'instruction **LINE (0,0)–(10,10),B** sera interprétée comme « tracer une ligne du point (0,0) au point (10,10) avec la couleur B ». Si on a utilisé une variable **B** dans le programme, sa valeur sera utilisée pour la couleur ; sinon

0 sera utilisé. Pour ne pas toucher à la couleur il faut ne pas utiliser l'espace réservé à la couleur, comme dans l'instruction `LINE (0,0)-(10,10),,B` dans laquelle la couleur 3 sera utilisée.

La figure 2.3 montre un écran qui a été rempli par des rectangles à l'aide du programme 2.4.

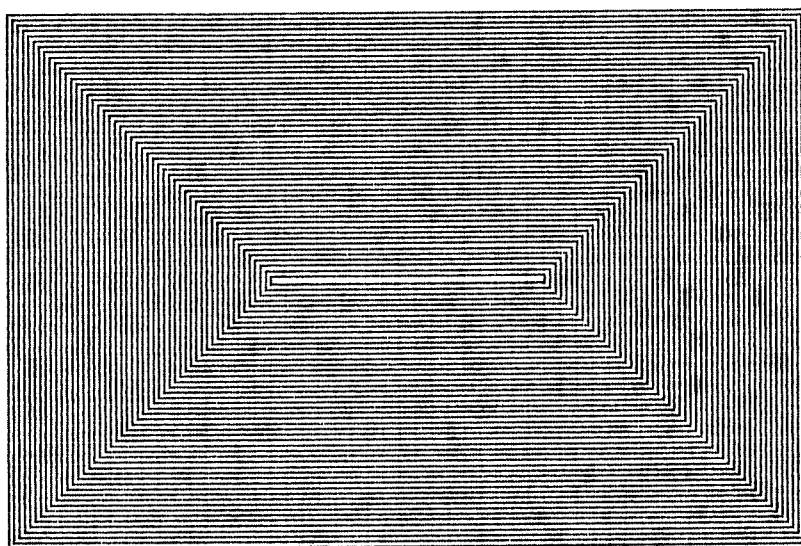


Fig. 2.3

```
0 2-4 : Fill with rectangles
10 SCREEN 1:CLS
20 FOR I=199 TO 100 STEP -2
30   LINE(199-I,199-I)-(I,I),3,B
40 NEXT
```

PROGRAMME 2.4

La plupart des figures de ce livre sont entourées d'un cadre pour indiquer les bords de l'écran, ce cadre est réalisé avec l'instruction `LINE(0,0)-(319,199),3,B`. (Cette instruction ne figure pas dans le programme.)

Les rectangles produits par l'instruction `LINE` ont leurs côtés parallèles aux axes des X et Y. Au chapitre 8 nous apprendrons comment tracer des rectangles ayant d'autres orientations.

2.4 LE DESSIN DE RECTANGLES PLEINS

On peut utiliser l'instruction `LINE` pour tracer des rectangles pleins (qu'on appellera aussi « boîtes ») et qui sont des rectangles colorés avec la

couleur du cadre. La syntaxe de cette instruction est LINE (*colonne 1, ligne 1*)—(*colonne 2, ligne 2*), *couleur*, BF. Comme pour les formats précédents, le LRP est (*colonne 2, ligne 2*). Le programme 2.5 remplit une partie de l'écran avec des carrés se chevauchant comme le montre la figure 2.4.

```
0  "" 2-5 : Overlapping squares ""
10 CLS:C=1
20 FOR I=0 TO 100 STEP 5
30   LINE (199-I,199-I)—(I,I),C,BF:C=C+1
40   IF C=4
       THEN
         C=1
50 NEXT
```

PROGRAMME 2.5

Une boîte de la taille de l'écran dans la couleur du fond produite par l'instruction LINE(0,0)—(319,199),0,BF est équivalente à CLS, sauf qu'elle laisse le LRP à (319,199), mais on peut également l'utiliser pour colorer l'écran d'une couleur autre que le fond. Par exemple, l'instruction LINE(0,0)—(319,199),2,BF, efface l'écran et le colore avec la couleur 2. Aux paragraphes 7.16 et 7.17 nous étudierons comment colorer certaines zones de textes de l'écran au moyen des boîtes.

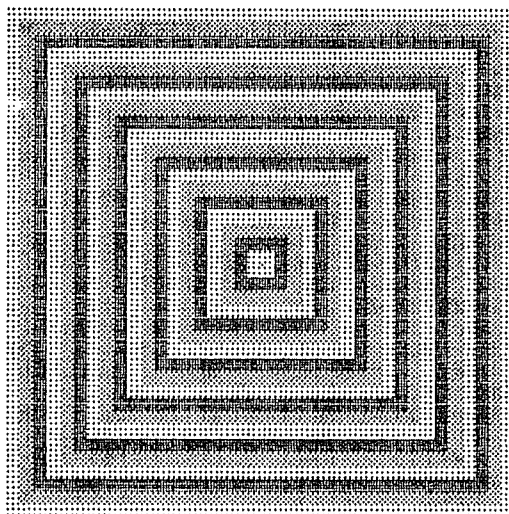


Fig. 2.4

2.5 LE TRACE D'UNE FONCTION AVEC UNE LIGNE CONTINUE

A l'aide de l'instruction LINE, il est possible de tracer une fonction avec une ligne continue, contrairement à la ligne en pointillé du programme 1.5. On peut faire cela si, au lieu de tracer simplement les points, on connecte le premier point avec le second, le second avec le troisième et ainsi de suite.

CLS laisse le LRP à (160,100). Comme nous allons connecter chaque point calculé avec le LRP le programme devra tracer le premier point ou sinon la première ligne partira du centre de l'écran. A la ligne 40 du programme, si $X = 0$ (ce qui est le cas pour le premier point) un point unique est affiché, si X est différent de zéro, une ligne sera tracée entre le dernier point tracé et le nouveau X, Y .

```
0  '° 2-6 : Function with continuous line °
10 SCREEN 1:CLS
20 FOR X=0 TO 319
30   Y=100+80°SIN(X°.04)
40   IF X=0
       THEN
         PSET (X,Y)
       ELSE
         LINE -(X,Y)
50 NEXT
```

PROGRAMME 2.6

On peut voir le graphe résultat de ce programme à la figure 2.5. Comparez-le avec celui de la figure 1.8.

Si on change la ligne 40 en :

```
40   IF X>0
       THEN
         LINE (PX,PY)-(X,Y)
```

et qu'on ajoute la ligne suivante :

```
45   PX=X:PY=Y
```

le graphe ne changera pas, mais cette méthode permettra un meilleur contrôle, comme on le verra aux paragraphes 2.5.1, 2.5.4 et d'autres.

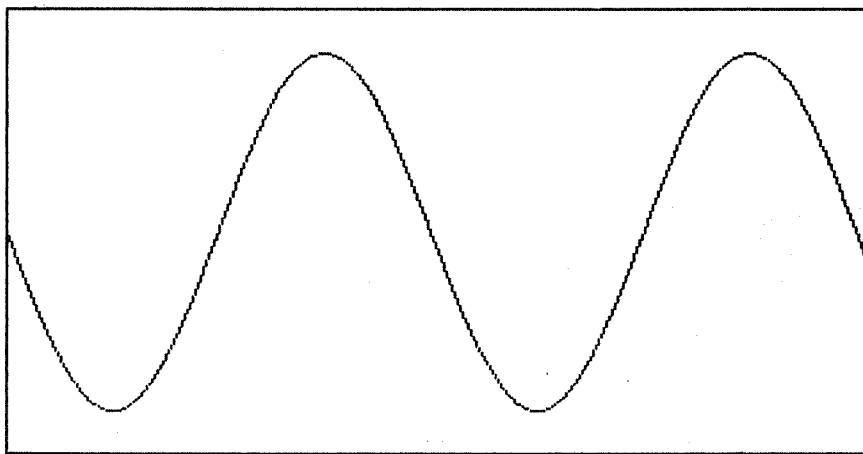


Fig. 2.5

2.5.1 La parabole

La forme la plus simple de l'équation² d'une parabole est

$$y = x^2$$

Cette fonction croît si vite que, si nous essayons de la tracer, on ne pourra avoir que quelques points sur l'écran. Encore pire, elle donne des valeurs si petites ou si grandes que quand on les utilise dans PSET ou LINE: elles entraînent une erreur de dépassement de capacité. Pour éviter cela il faut rééchelonner Y en plus petit. Dans le programme 2.7, Y est divisé par 130, ce qui fait que la totalité de la parabole tient sur l'écran, comme on le voit à la figure 2.6. Les lignes 20 et 30 tracent les axes des X et Y.

```
0  *** 2-7 : Parabola **
10 SCREEN 1:COLOR ,0:CLS
20 LINE(0,199)-(319,199)
30 LINE(160,0)-(160,199)
40 FOR X=-160 TO 160
50   Y=X*X/130
60   IF X=-160
      THEN
        PSET(160+X,199-Y)
      ELSE
        LINE-(160+X,199-Y)
70 NEXT
```

PROGRAMME 2.7

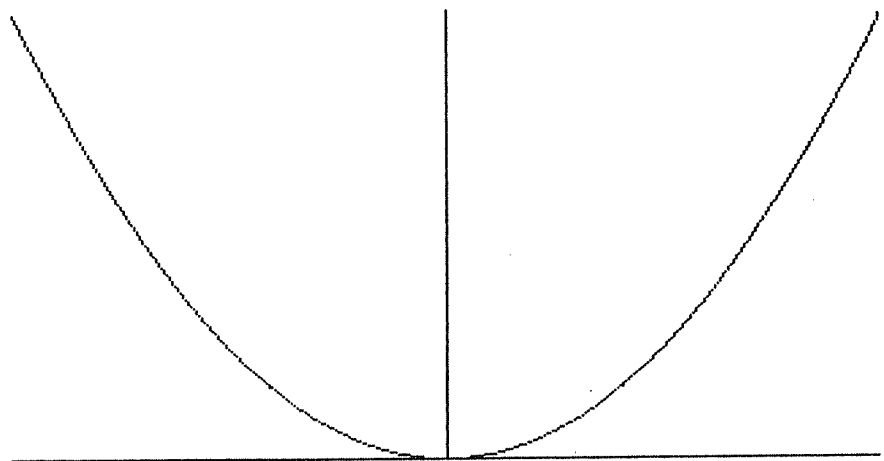


Fig. 2.6

Une autre façon pratique de tracer une fonction est de balayer l'écran de chaque point (X,Y) au point $(X,199)$, c'est-à-dire tracer une ligne jusqu'au bas de l'écran. Ceci fait apparaître la fonction qui sépare clairement les parties qui sont en dessus et en dessous d'elle. Si on change la ligne 60 du programme en :

```
60 LINE(160+X,199-Y)-(160+X,199)
```

non seulement la fonction sera tracée mais également la zone située en dessous comme le montre la figure 2.7.

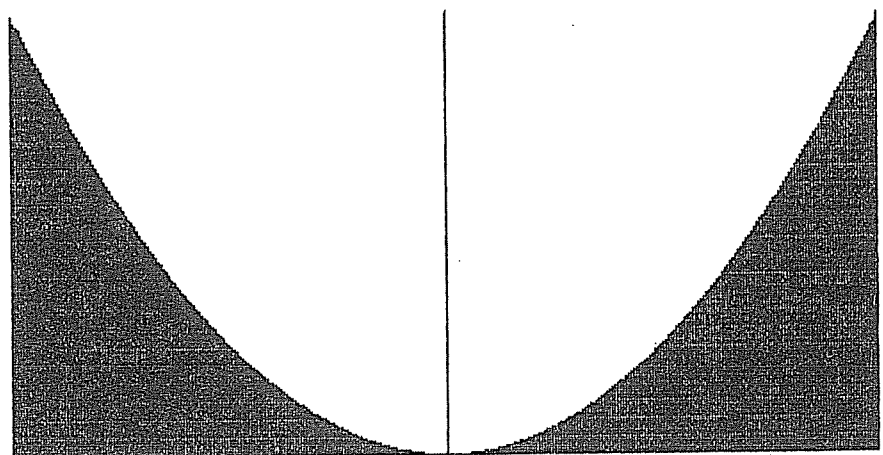


Fig. 2.7

Une autre amélioration peut consister à changer la couleur tous les n points pour visualiser des intervalles sur l'axe des X . Si on change la ligne 60 du programme 2.7 en :

```
60  LINE(160+X,199-Y)-(160+X,199),1+ABS(X)/10 MOD 3
```

la zone balayée par la parabole changera de couleur chaque 10 pixels comme le montre la figure 2.8. Il faut utiliser la valeur absolue de X , parce qu'un code couleur négatif n'est pas autorisé.

2.5.2 Un programme général pour tracer les fonctions

Comme on a souvent besoin de tracer des fonctions, c'est une bonne idée que de disposer d'un programme dans lequel l'utilisateur n'aie pas à se préoccuper de l'échelle et de la modification des paramètres. La méthode pour tracer les points ou les lignes qui composeront la fonction sera toujours la même ; c'est l'amplitude de la fonction qui complique les choses. Il y a des fonctions qui ont un comportement intéressant dans certains petits intervalles, et il y en a d'autres qui croissent si vite ou si lentement que si on n'utilise pas une échelle appropriée pour l'axe des X et des Y , la partie visualisée est peu utile, si ce n'est complètement inutile.

Dans le programme 2.8, la fonction à tracer est définie à la ligne 10 ; chaque fois qu'on veut une nouvelle fonction, on n'a seulement qu'à

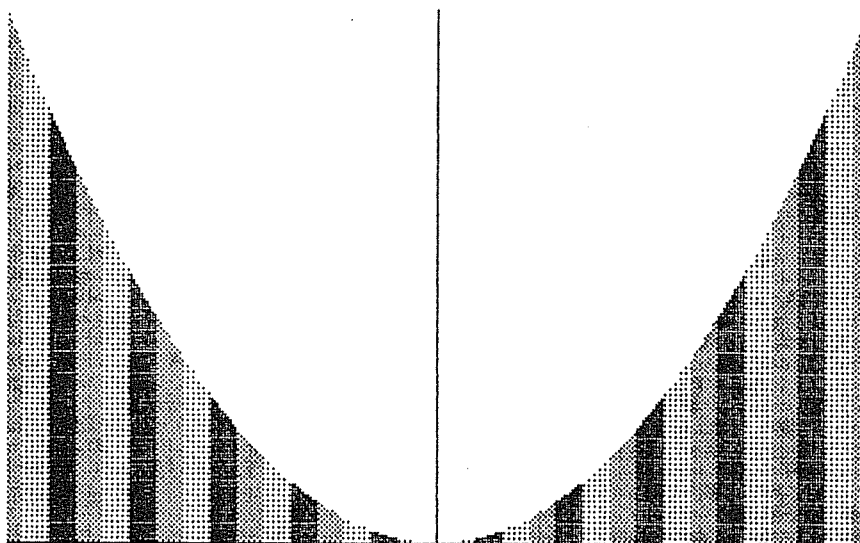


Fig. 2.8

changer cette ligne. Nous allons résoudre le problème de l'amplitude de la fonction en demandant les limites désirées pour l'axe des X et des Y. Ceci nous permettra d'étudier n'importe quelle partie de la fonction, quelle soit grande ou petite. Si les limites sont de -1000 à 1000 pour l'axe des X, et de -1000 à 1000 pour l'axe des Y par exemple, on visualisera une grande partie de la fonction, bien qu'on perdra de petits détails. Si les limites sont de -0,1 à 0,1 pour chacun des deux axes, on ne visualisera qu'une petite partie de la fonction, mais avec précision.

Le programme 2.8 demande, aux lignes 10 et 20, les limites pour les axes des X et des Y. La différence $X2-X1$ (point d'arrivée moins point de départ) est la longueur de l'intervalle sur lequel la fonction sera tracée. Si on divise cette différence par 319 (dans le programme cette valeur est affectée à SX) nous aurons déterminé le pas de X puisqu'il y a exactement 320 pas entre $X1$ et $X2$. On utilisera la variable FX pour calculer chaque valeur Y. Ainsi à la ligne 80 elle est fixée à la valeur de $X1$. Une boucle FOR (avec des valeurs de 0 à 319) sera utilisée pour traverser l'écran horizontalement. A chaque passage dans la boucle, on ajoutera SX à FX, ainsi quand $X = 319$, FX sera égal à $X2$.³

Les équations sont :

$$\begin{aligned} D &= X2 - X1 \\ SX &= D / 319 \\ FX &= X1 + I * SX \end{aligned}$$

Quand Y est égal à $Y1$, il doit être tracé en bas de l'écran ; ceci peut être réalisé si on retranche $Y1$ de chaque Y. Si on divise la longueur de l'intervalle $Y(Y2-Y1)$ par 199 et qu'on multiplie le résultat par Y (après en avoir retranché $Y1$), alors quand Y sera égal à $Y2$, ce point sera tracé en haut de l'écran et chaque point dans l'intervalle sera échelonné en conséquence.

Les équations sont :

$$\begin{aligned} D &= Y2 - Y1 \\ F &= 199 - D \\ YF &= (Y - Y1) * F \end{aligned}$$

où YF est la valeur utilisée en fait sur l'écran.

```
0  ** 2-8 : Plot a function in the chosen ranges **
5  DEF FN A(X)=SIN(X)
6  SCREEN 1:CLS
10 INPUT"x1,x2 ";X1,X2:IF X2<X1 THEN 10
20 INPUT"y1,y2 ";Y1,Y2
25 CLS
30 SX=(X2-X1)/319:SY=199/(Y2-Y1):FX=X1
40 FOR X=0 TO 319:
    Y=(FN A(FX)-Y1)*SY:PSET(X,199-Y):FX=FX+SX:
NEXT
```

PROGRAMME 2.8

2.5.3 Entrée de la fonction

L'inconvénient des programmes de tracé de fonctions étudiés jusqu'ici est que quand on veut passer à une fonction différente, le programme doit être modifié. Si le programme demande et obtient la fonction comme une chaîne, il est très difficile d'évaluer l'expression pour calculer les valeurs qui lui correspondent. Heureusement il y a un moyen pour laisser BASIC accomplir ce travail, en l'écrivant sur le disque comme une ligne de programme et en faisant CHAIN MERGE entre le fichier créé et le programme en mémoire. Ainsi la fonction devient une partie du programme, et peut être utilisée directement. Dans le programme 2.9 la ligne 20 rentre la chaîne. Si la chaîne est vide, l'exécution du programme se poursuit à la ligne 60. Cette option permet des exécutions répétées avec une fonction rentrée préalablement mais avec des limites différentes. Quand la chaîne n'est pas vide, une image sur disque de la fonction est créée sur disque aux lignes 30 et 40. Si la chaîne rentrée est par exemple EXP(X), la ligne

```
60 DEF FN A(X)=EXP(X)
```

est écrite sur disque. la ligne 50 fusionne (CHAIN MERGE) cette ligne avec le programme, et l'exécution continue. A partir de ce point, la définition de fonction de la ligne 60 devient une partie régulière du programme.

```
0 ' ** 2-9 : Plot a function entered as a string **
10 SCREEN 1:CLS
20 LINE INPUT "F(X)=";W$:
   IF W$=""
   THEN
     60
30 W$="60 DEF FN A(X)="+W$
40 OPEN "O",1,"DUMMY.BAS":PRINT#1,W$:CLOSE
50 CHAIN MERGE "DUMMY.BAS",60,ALL
60 DEF FN A(X)=100*SIN(X)*SIN(X*.5)*SIN(X*.25)
70 INPUT "x1,x2 ";X1,X2:
   IF X2<=X1
   THEN
     70
80 INPUT "y1,y2 ";Y1,Y2:
   IF Y2<=Y1
   THEN
     80
90 CLS
100 SX=(X2-X1)/319:SY=199/(Y2-Y1):FX=X1
110 FOR X=0 TO 319
120   Y=(FN A(FX)-Y1)*SY
130   IF X=0
   THEN
     PSET (X,199-Y)
   ELSE
     LINE-(X,199-Y)
```

```

140   FX=FX+SX
150 NEXT

```

PROGRAMME 2.9

2.5.4 La fonction cercle

Les cercles étant des courbes très importantes, nous allons étudier maintenant comment les générer avec des fonctions. Mathématiquement un cercle a la caractéristique que la distance de chaque point (X,Y) au centre est égale à une constante appelée le rayon. Ce fait est énoncé dans l'équation 2.1 :

$$r^2 = x^2 + y^2 \quad (2.1)$$

Nous prendrons X comme variable indépendante (ainsi nous pourrons l'utiliser dans une boucle FOR), et à l'aide de l'équation 2.1 nous calculerons les valeurs de Y. Pour faire cela nous déplaçons Y à gauche du signe égale et tous les autres termes sur la droite, et on a l'équation 2.2 :

$$y = \pm \sqrt{r^2 - x^2} \quad (2.2)$$

La partie positive forme la moitié supérieure du cercle et la partie négative forme la moitié inférieure. A l'aide de l'équation 2.2 le programme 2.10 dessine la partie supérieure d'un cercle de rayon 90. A la ligne 40 on obtient la valeur de Y en prenant la racine carrée de

$$r^2 - x^2$$

Notez que dans le programme 2.10 nous utilisons R*R pour calculer le carré de R au lieu de R↑2, de même X*X au lieu de X↑2. Ceci parce que la multiplication est calculée plus vite que l'exponentiation en BASIC. A la ligne 50 nous ajoutons 160 à chaque valeur X ainsi le point pour lequel X = 0 est placé au centre de l'écran, et le cercle est déplacé de 160 pixels sur la droite.

X ne peut prendre que des valeurs entre -R et +R, ou sinon (R*R -X*X) deviendrait négatif. La fonction SQR n'est définie que pour des valeurs positives.

```

O "" 2-10 : Semi-circle with equation ""
10 SCREEN 1:CLS
20 R=90
30 FOR X=-R TO R
40   Y=SQR(R*R-X*X)

```

```

50 PSET(X+160,Y)
60 NEXT

```

PROGRAMME 2.10

La figure 2.9 montre la moitié de cercle tracée par le programme 2.10.

Afin de dessiner la partie supérieure du cercle on peut utiliser les valeurs calculées à la ligne 40 une deuxième fois. Comme le cercle est parfaitement symétrique par rapport à l'axe des Y, pour chaque point (X,Y) appartenant au cercle le point (X,-Y) appartiendra également au cercle⁴. Le programme 2.11 ajoute 100 à Y à la ligne 50 pour avoir le point pour lequel Y = 0 situé au centre du cercle et à la ligne 60 on retranche 100 à Y, de sorte qu'une image miroir du cercle est produite à partir de Y = 100 et au-dessus. La figure 2.10 montre le cercle résultant.

```

0  " 2-11 : Circle with equation "
10 SCREEN 1:CLS
20 R=90
30 FOR X=-R TO R
40   Y=SQR(R*R-X*X)
50   PSET(X+160,100+Y)
60   PSET(X+160,100-Y)
70 NEXT

```

PROGRAMME 2.11

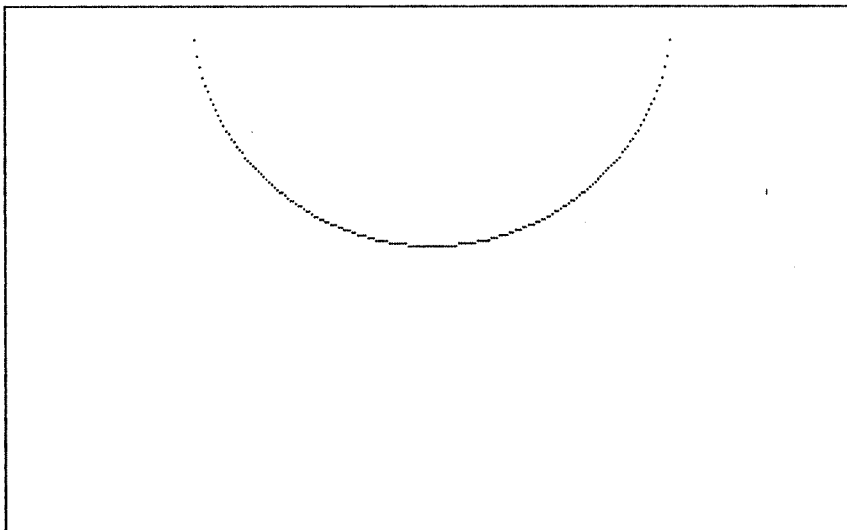


Fig. 2.9

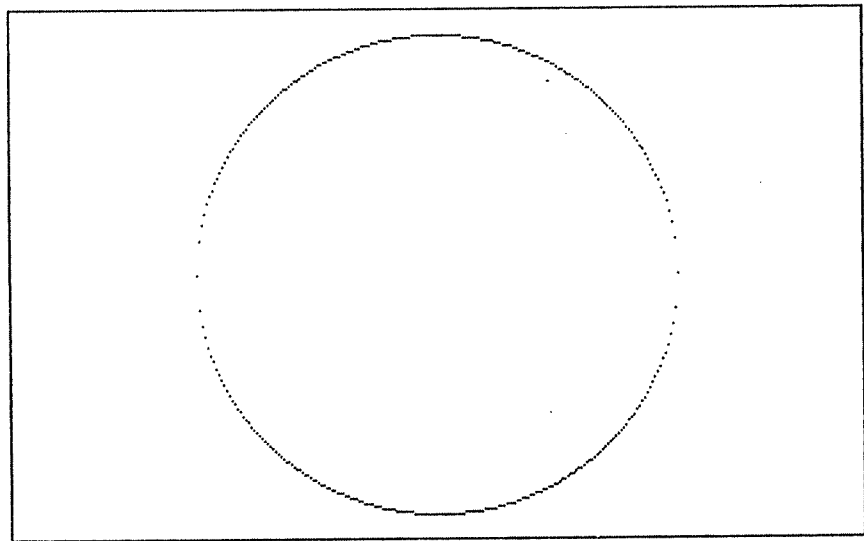


Fig. 2.10

Pour ajouter de la souplesse à la création d'un cercle, le programme 2.12 demande les coordonnées du centre, le rayon et le code couleur. Le cercle est tracé ici avec une ligne continue comme le montre la figure 2.11 et où $XC = 200$, $YC = 110$, $R = 85$ et $C = 3$.

```

0  *** 2-12 : Circle with variable parameters *
10 SCREEN 1:CLS
20 INPUT"Center (X,Y) ";XC,YC:INPUT"Radius ";R
25 INPUT"Color (0-3) ";C:CLS
30 FOR X=-R TO R
40   Y=SQR(R*R-X*X)
50   IF X>-R
       THEN
           LINE(PX+XC,YC+PY)-(X+XC,YC+Y),C:
           LINE(PX+XC,YC-PY)-(X+XC,YC-Y),C
60   PY=Y:PX=X
70 NEXT

```

PROGRAMME 2.12

2.6 METHODES PRIMAIRES DE TRACE DE DROITES

Dans ce paragraphe nous allons étudier deux techniques pour tracer des lignes droites sans utiliser l'instruction LINE. Bien que le fait de tracer

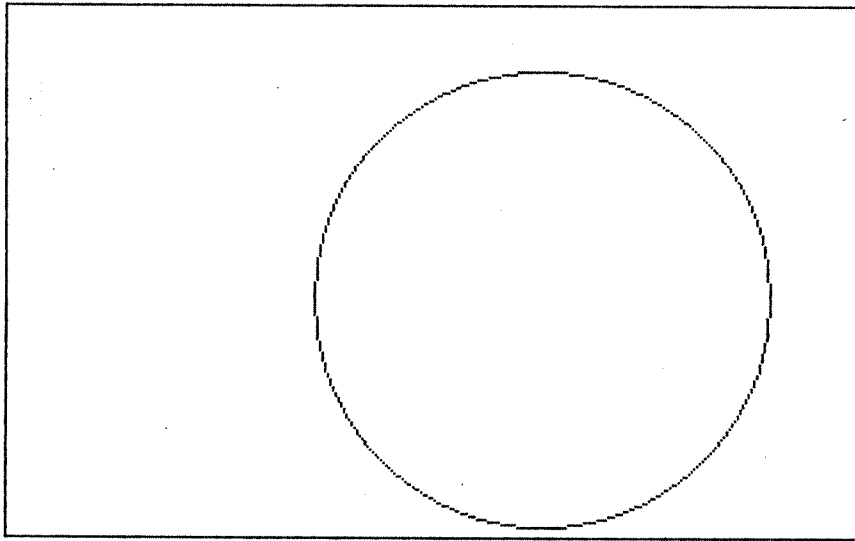


Fig. 2.11

des droites avec cette méthode soit long et compliqué, le contrôle acquis nous permet l'utilisation de certaines techniques qu'il est impossible d'appliquer à l'instruction LINE.

2.6.1 Point par point

Nous allons tracer une ligne entre le point P1 (de coordonnées X1, Y1) et P2(X2,Y2) comme le montre la figure 2.12. Nous déterminons la distance horizontale entre P1 et P2 que nous appelons DX en faisant l'opération $DX = X2 - X1$. Nous déterminons la distance verticale entre P1 et P2 que nous appelons DY en faisant l'opération $DY = Y2 - Y1$ et nous déterminons la distance entre P1 et P2 que nous appelons D en faisant soit l'opération $D = \text{SQR}((X2-X1)^2 + (Y2-Y1)^2)$ soit l'opération $D = \text{SQR}(DX*DX + DY*DY)$. Nous déterminerons ensuite un vecteur unitaire, c'est-à-dire que les incréments sur les axes des X et des Y ajoutés D fois iront exactement du point P1 au point P2. Si nous divisons DX par D nous obtiendrons la longueur (que nous appellerons SX) qui ajoutée D fois sera égale à DX, ce qui fait que nous obtiendrons l'incrément dont nous avons besoin dans le sens des X. L'équivalent dans le sens des Y (que nous appellerons SY) est obtenu en divisant DY par D. Et maintenant une simple boucle FOR tracera la ligne :

```
FOR I=0 TO D
  PSET(X1+SX*I,Y1+SY*I)
NEXT
```

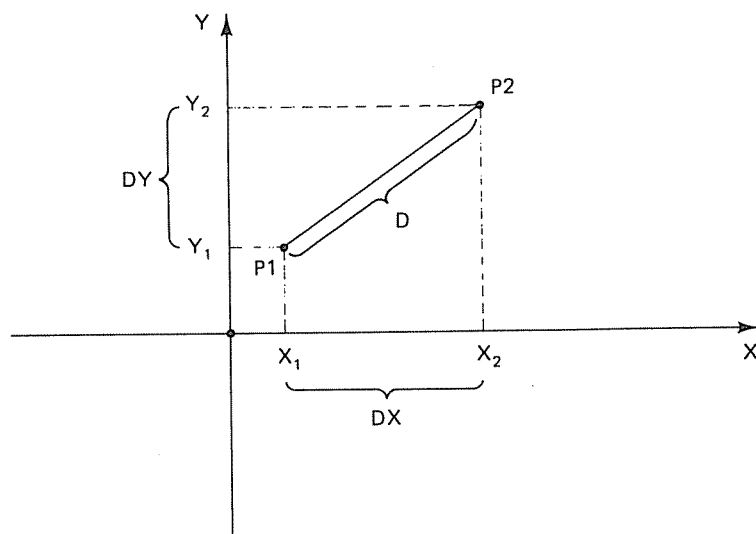


Fig. 2.12

Le programme 2.13 trace une ligne entre les points dont les coordonnées sont rentrées aux lignes 20 et 30.

```
O °° 2-13 : Line between two points °°
10 SCREEN 1:COLOR 0,0:CLS
20 INPUT"First point (X,Y) ";X1,Y1
30 INPUT"Second point (X,Y) ";X2,Y2
40 DX=X2-X1:DY=Y2-Y1
50 D=SQR(DX*DX+DY+DY)
60 SX=DX/D:SY=DY/D
70 FOR I=0 TO D
80   PSET(X1+SX*I,Y1+SY*I)
90 NEXT
```

PROGRAMME 2.13

Les lignes avec des couleurs changeantes. A la ligne 80 du programme 2.13, tous les points étaient tracés avec la même couleur, mais rien ne nous empêche de la changer périodiquement pour créer une ligne de plusieurs couleurs. Si on change la ligne 80 en

```
80 PSET(X1+SX*I,Y1+SY*I),1+I/S MOD 4
```

la ligne changera de couleur tous les S points.

Ligne en pointillé. Une autre possibilité pour tracer les lignes avec cette méthode est d'espacer les points pour rendre la ligne pointillée et non continue. Si on change la ligne 70 du programme 2.13 en

```
70 FOR I=0 TO I STEP S
```

la ligne sera continue si $S \leq 1$, et en pointillé quand $S > 1$. La figure 2.13 montre quatre lignes tracées avec $S = 1$, $S = 2$, $S = 4$ et $S = 6$.

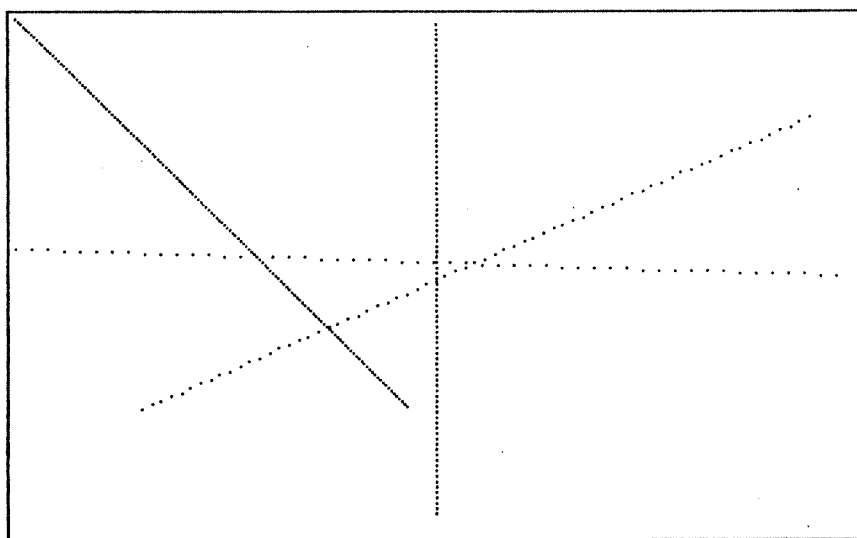


Fig. 2.13

2.6.2 Equation d'une droite

Chaque ligne droite peut être décrite par une équation² de la forme

$$f(x) = mx + b$$

où m est la pente (ou inclinaison) de la droite par rapport à l'axe des X et b (appelée habituellement ordonnée à l'origine) est la coordonnée Y du point où la droite croise l'axe des Y . La droite décrite par l'équation

$$y = x$$

par exemple, a en fait la forme

$$y = (1 \times x) + 0$$

C'est-à-dire que la pente est 1 et que la droite coupe l'axe des Y au point 0. L'équation

$$y = (0.3 \times x) + 1$$

décrit la droite qui coupe l'axe des Y au point 1 (au point (0,1)) et a une pente de 0,3.

La pente d'une droite est la tangente de l'angle que cette droite forme avec l'axe des X. La figure 2.14 montre la droite tracée à partir de l'équation suivante :

$$y = (2 \times x) - 3$$

Droite définie par son équation. Quand on peut avoir l'équation de la droite, la tracer peut être fait aussi facilement que de tracer n'importe quelle fonction. Le programme 2.14 trace la droite décrite par la fonction dont l'équation est définie à la ligne 20.

```
0  "" 2-14 : Line with equation ""
10 SCREEN 1:CLS
20 DEF FN LN(X)=.3*X+1
30 FOR X=0 TO 319
40   PSET(X,199-FN LN(X) )
50 NEXT
```

PROGRAMME 2.14

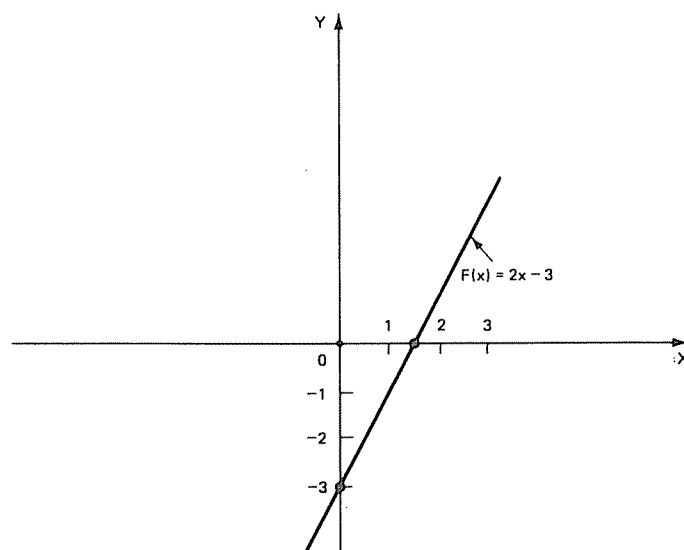


Fig. 2.14

Comme nous traçons une fonction, il faut utiliser la méthode expliquée au paragraphe 2.5 si l'on veut tracer la ligne en continu. On ne peut pas décrire les lignes verticales avec cette méthode parce que la pente serait infinie et il n'y aurait soit aucune intersection (une ligne verticale avec $X \neq 0$), soit une infinité d'intersections (une ligne verticale avec $X = 0$). Dans ces cas on peut tracer la droite avec la boucle :

```
30 FOR Y=Y1 TO Y2
40   PSET(X1,Y)
50 NEXT
```

Calcul de l'équation d'une droite. Quand on nous donne deux points d'une droite, on calcule son équation ainsi :

1. On calcule la pente en divisant la distance verticale entre les deux points ($Y2 - Y1$) par la distance horizontale ($X2 - X1$), ainsi

$$m = (Y2 - Y1) / (X2 - X1)$$

2. Le point d'intersection est calculé par la formule

$$b = (-m \times X1) + Y1.$$

Exemple 1.

L'équation de la ligne définie par les points (5,5) et (45,35) est :

$$m = (35 - 5) / (45 - 5) = 30 / 40 = 0,75$$

$$b = -0,75 \times 5 + 5 = 1,25$$

$$f(x) = 0,75 \times x + 1,25$$

Exemple 2.

L'équation de la ligne définie par les points (100,100) et (20,30) est :

$$m = (30 - 100) / (20 - 100) = -70 / -80 = 0,875$$

$$b = -0,875 \times 100 + 100 = 12,5$$

$$f(x) = 0,875 \times x + 12,5$$

Exemple 3.

L'équation de la droite définie par les points (160,10) et (160,100) ne peut pas être calculée par la méthode expliquée ci-dessus parce que $X1 = X2$. La droite peut être tracée avec la boucle.

```
100 FOR Y=10 TO 100
110   PSET(160,Y)
120 NEXT
```

2.6.3 Ecrêtage des extrémités

Quand un ou les deux points qui déterminent une droite sont en dehors de l'écran, BASIC prend comme coordonnées les valeurs autorisées les plus proches. Ceci conduit en général à une déformation de la fonction d'origine. Le programme 2.15 dessine une maison avec un toit élevé. En rentrant différentes valeurs à la ligne 20, la maison est dessinée à des hauteurs différentes. Quand on rentre des valeurs négatives, le toit est déformé. Quand $D = -50$ par exemple, la ligne gauche du toit qui devrait être dessinée de (120,150) à (160,-50) est dessinée de (120,50) à (160,0).

O °° 2-15 : Shows distortion of houses °°

```
10 SCREEN 1:CLS
20 INPUT "Height ";D
30 CLS:GOSUB 100
40 W$=INPUT$(1)
50 GOTO 20
100 LINE(120,100+D)-(200,100+D)
110 LINE(120,100+D)-(120,150+D)
120 LINE(120,150+D)-(200,150+D)
130 LINE(200,150+D)-(200,100+D)
140 LINE(120,100+D)-(160,0+D)
150 LINE(160,0+D)-(200,100+D)
160 RETURN
```

PROGRAMME 2.15

La figure 2.15 montre trois maisons, dont deux ont des sections qui ne s'intègrent pas entièrement à l'écran et sont donc déformées.

Nous allons présenter ici deux méthodes pour tracer la partie de la ligne qui se trouve à l'intérieur de l'écran sans la déformation causée par l'instruction LINE et nous décrirons brièvement une troisième méthode à l'exercice 2.4.

Par rapport à l'écran il y a quatre types de droites comme le montre la figure 2.16.

La ligne A est complètement à l'intérieur de l'écran.

La ligne B a un des points qui la définit hors de l'écran, l'autre à l'intérieur.

La ligne C a ses deux points qui la définissent hors de l'écran mais une partie de la ligne traverse l'écran.

La ligne D est complètement hors de l'écran.

D'après la liste ci-dessus, il est clair qu'on n'obtient rien de plus en vérifiant tout simplement si les points sont ou ne sont pas à l'intérieur

de l'écran ; les deux peuvent être à l'extérieur et une partie de la ligne traverser l'écran.

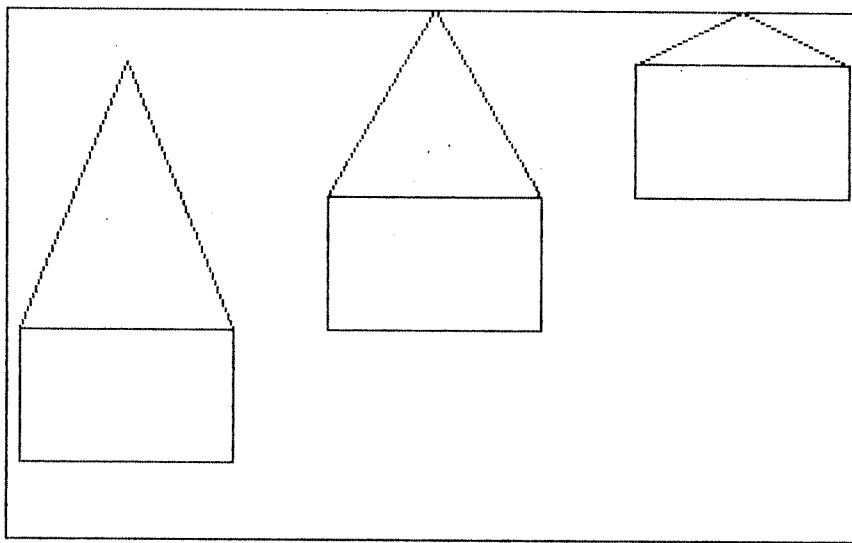


Fig. 2.15

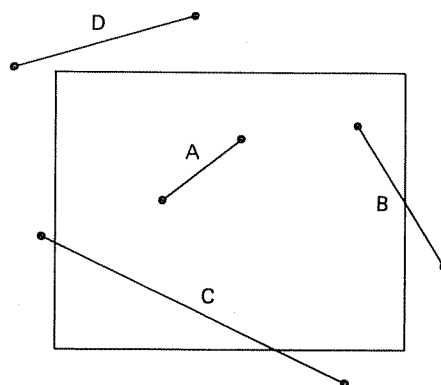


Fig. 2.16

Dans la première méthode pour éviter cette déformation, on utilise la méthode point par point pour tracer les droites, comme nous l'avons décrit au paragraphe 2.6.1. Comme chaque point est calculé individuellement le problème de la déformation est complètement évité. Toutefois le processus peut être lent. Si la ligne est très longue (peut-être la majeure partie de celle-ci se situe-t-elle en dehors de l'écran) le processus peut sembler interminable. Le cas extrême, la droite de $(-32767, -32767)$ à $(32767, 32767)$ prend à peu près 15 minutes pour s'accomplir. Un autre

problème est que les points avec $320 \leq X \leq 639$ et $0 \leq Y \leq 197$ sont tracés par erreur à l'intérieur de l'écran, comme il est décrit au paragraphe 1.8. Ce problème peut être résolu facilement en rejetant ces points, mais faire cela ralentit le processus encore plus, du fait qu'il faut maintenant tester chaque point avant de le tracer.

En dépit de ces problèmes, quand on s'attend à ce que les lignes soient « à peu près normales » on peut utiliser cette méthode. Le programme 2.16 utilise cette méthode pour dessiner la maison du programme 2.15.

```
0  ' ** 2-16 : Clip lines dot by dot **
10 SCREEN 1:CLS
20 INPUT "Height ";HT
30 FOR I=1 TO 6
40   READ X1,Y1,X2,Y2
60   Y1=Y1+HT:Y2=Y2+HT
62   DX=X2-X1:DY=Y2-Y1
64   D=SQR(DX*DX+DY*DY)
66   SX=DX/D:SY=DY/D
70   FOR J=0 TO D
72     PSET (X1+SX*J,Y1+SY*J)
74   NEXT
110 NEXT
120 END
1000 DATA 120,100,200,100
1010 DATA 120,100,120,150
1020 DATA 120,150,200,150
1030 DATA 200,150,200,100
1040 DATA 120,100,160,0
1050 DATA 160,0,200,100
```

PROGRAMME 2.16

La figure 2.17 montre les maisons dessinées par le programme 2.16 ; remarquez la différence avec la figure 2.16.

La seconde méthode met en œuvre trois étapes :

1. Si les deux points sont à l'intérieur de l'écran, la ligne peut être tracée directement.
2. Si le premier point est à l'extérieur de l'écran, on utilise la méthode point par point pour calculer les points de la droite. Dès qu'on trouve un point à l'intérieur de l'écran, les coordonnées du premier point sont remplacées par celles du point qu'on vient de trouver. Si on atteint le second point et qu'on n'a trouvé aucun point à l'intérieur de l'écran, c'est que la ligne est complètement en dehors de l'écran.
3. Si le second point est en dehors de l'écran, on utilise la méthode point par point (avec les valeurs pour la boucle FOR inversées pour partir du point 2 et avancer vers le point 1) pour localiser le premier point à l'intérieur de l'écran. Si on en trouve un, les coordonnées du second point sont remplacées par celles du point trouvé.

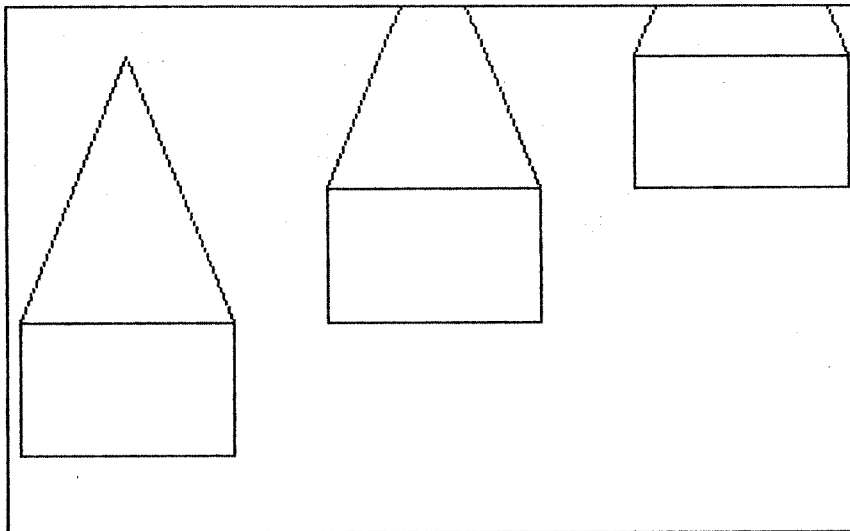


Fig. 2.17

Le sous-programme de la ligne 7500 du programme 2.17 utilise cette méthode pour écrêter les segments extérieurs de la maison du programme 2.15.

```

0  ' ** 2-17 : Clip lines by finding points **
1  ' ** that are in the line and inside the screen **
10 SCREEN 1:CLS
20 INPUT "Height ";D:CLS
30 FOR I=1 TO 6
40   READ X1,Y1,X2,Y2
50   XX1=X1:XX2=X2
60   YY1=Y1+D:YY2=Y2+D
70   GOSUB 7500
110 NEXT
120 END
2140 DATA 120,100,200,100
2150 DATA 120,100,120,150
2160 DATA 120,150,200,150
2170 DATA 200,150,200,100
2180 DATA 120,100,160,0
2190 DATA 160,0,200,100
7500 ' ** Clip line **
7505 XT1=XX1:XT2=XX2:YT1=YY1:YT2=YY2
7510 IF XX1<0 OR XX1>319 OR YY1<0 OR YY1>199
    THEN
        P1=-1: ' Point is outside the screen
7520 IF XX2<0 OR XX2>319 OR YY2<0 OR YY2>199
    THEN
        P2=-1: ' Point 2 is outside the screen
7530 DX=XX2-XX1:DY=YY2-YY1
7540 DD=SQR(DX*DX+DY*DY)
7550 SX=DX/DD:SY=DY/DD
7560 IF NOT P1 AND NOT P2

```

```

      THEN
        7670
7570 IF NOT P1
      THEN
        7630
7580 FOR II=0 TO DD
7590   XT=XX1+SX*II:YT=YY1+SY*II
7600   IF XT>=0 AND XT<=319 AND YT>=0 AND YT<=199
      THEN
        XX1=XT:YY1=YT:GOTO 7620
7610 NEXT:RETURN: ' Line is outside the screen
7620 IF NOT P2
      THEN
        7670
7630 FOR II=DD TO 0 STEP -1
7640   XT=XT1+SX*II:YT=YT1+SY*II
7650   IF XT>=0 AND XT<=319 AND YT>=0 AND YT<=199
      THEN
        XX2=XT:YY2=YT:GOTO 7670
7660 NEXT:RETURN: 'Line is outside the screen
7670 LINE(XX1,YY1)-(XX2,YY2):RETURN

```

PROGRAMME 2.17

2.7 REPRESENTATION VECTORIELLE DES FIGURES

Une manière pratique pour dessiner les objets est d'utiliser les coordonnées de tous leurs points, de même que les informations concernant la connexion des points entre eux. Par exemple un programme pour tracer un carré est facile à écrire en utilisant les coordonnées des points limites directement. Cependant, pour écrire un programme qui dessine un triangle, il faut pratiquement tout refaire. Avec la représentation vectorielle, un ensemble de données définit les points et un second ensemble définit les connexions de ces points entre eux. En changeant tout simplement les données, on peut utiliser un même programme pour tracer des figures complètement différentes.

Au programme 2.18, on utilise les tableaux X et Y pour mémoriser les coordonnées de chaque point d'une figure et LO et LD pour mémoriser les numéros des points d'origine et de destination des lignes. Le rectangle dessiné par l'instruction LINE (30,30)-(100,100),3,B aura, en représentation vectorielle, les points et les lignes de la table 2.1. Par exemple, la ligne 4 dit que le point 4 (100,30) est connecté au point 1 (30,30). C'est l'équivalent de la commande LINE (100,30)-(30,30).

Table 2.1 (a)

Point	X	Y
1	30	30
2	30	100
3	100	100
4	100	30

Table 2.1 (b)

<i>Ligne</i>	<i>Origine</i>	<i>Destination</i>
1	1	2
2	2	3
3	3	4
4	4	1

Il ne semble pas qu'un rectangle tire beaucoup d'avantages de cette forme de représentation. (En fait, cela semble beaucoup plus compliqué qu'une seule instruction LINE.) Toutefois des formes plus complexes tireront avantage de cette méthode, non seulement dans la création, mais également dans la manipulation de ses composantes. Le programme 2.18 dessine l'objet dont les données sont dans les tableaux présentés ci-dessus. *NP* contient le nombre de points et *NL* le nombre de lignes. Il existe deux options pour rentrer les données : par le clavier ou à partir du disque. Ceci facilite la re-cr  ation de figures ayant un grand nombre de points. Une fois que la donn  e a   t   tap  e, elle est   crite sur disque sous le nom de fichier sp  cifi      la ligne 90. Elle peut ensuite   tre lue et utilis  e directement.

```

0  ' ** 2-18 : Draw a figure with **
1  '    ** vector representation **
10 SCREEN 1:CLS
15 DX=40:DY=12
20 DIM X(100),Y(100),LO(100),LD(100)
30 INPUT "1-Enter data. 2-Read from disk ";W$
   ON W GOTO 40,160
40 FOR I=1 TO 100
50   PRINT "Point #"I" (X) ";:INPUT W$
60   IF W$=""
       THEN
           NP=I-1:PRINT:GOTO 90
       ELSE
           X(I)=VAL(W$)
70   INPUT "          (Y) ";Y(I)
80 NEXT:NP=100
90 PRINT:INPUT "Filename ";FL$
100 OPEN "O",1,FL$:PRINT#1,NP:
   FOR I=1 TO NP:
       PRINT#1,X(I):PRINT#1,Y(I):
   NEXT:
   PRINT#1,NL:
   FOR I=1 TO NL:
       PRINT#1,LO(I):PRINT#1,LD(I):
   NEXT:CLOSE
110 FOR I=1 TO 100
120   PRINT "Line #"I" (origin)";:INPUT W$
130   IF W$=""
       THEN
           NL=I-1:GOTO 180

```

```

ELSE
  LO(I)=VAL(W$)
140 INPUT"      (destination)";LD(I)
150 NEXT:
  NL=100:GOTO 180
160 PRINT:INPUT"Filename ";FL$
170 OPEN"I",1,FL$:INPUT#1,NP:
  FOR I=1 TO NP:
    INPUT#1,X(I):INPUT#1,Y(I):
  NEXT:
  INPUT#1,NL:
  FOR I=1 TO NL:
    INPUT#1,LO(I):INPUT#1,LD(I):
  NEXT:CLOSE
180 CLS
190 FAC=7
200 FOR I=1 TO NL
210 LINE(DX+X(LO(I))*FAC,DY+Y(LO(I))*FAC)-
      (DX+X(LD(I))*FAC,DY+Y(LD(I))*FAC)
220 NEXT
230 W$=INPUT$(1):CLS

```

PROGRAMME 2.18

La figure 2.18 montre le dessin d'un avion sur un papier graphique, les données obtenues à partir de ce dessin apparaissent à la table 2.2.

La figure 2.19 montre le graphique réalisé à partir des données de la table 2.2.

Un des grands avantages de cette méthode vectorielle est que la mise en place et à l'échelle des figures est grandement simplifiée. Si on change la ligne 210 du programme 2.18 en

```

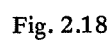
210 LINE (DX+X(LO(I))*FAC,DY+Y(LO(I))*FAC)-
      (DX+X(LD(I))*FAC,DY+Y(LD(I))*FAC),

```

l'objet peut être relocalisé sur l'écran en changeant simplement les variables DX et DY, et changé d'échelle en modifiant simplement la variable FAC. La figure 2.20 nous montre des avions de tailles différentes, dessinés avec les données de la table 2.2.

CONDITIONS LIMITES

Tous les paramètres des coordonnées des instructions LINE doivent être supérieurs ou égaux à -32768 et inférieurs ou égaux à 32767. Le code couleur doit être supérieur ou égal à zéro et inférieur à 256. Les codes couleur entre 4 et 255 produisent la dernière couleur tracée. Quand n'importe laquelle des coordonnées est en dehors de l'écran, la valeur autorisée la plus proche est utilisée à sa place. L'instruction LINE (0,0) - (32000,100) est en fait tracée comme LINE (0,0) - (319,100).



<i>Point</i>	X	Y
1	0	11
2	0	14
3	18	14
4	19	15
5	23	15
6	30	12.5
7	35	12.5
8	23	10
9	19	10
10	18	11
11	7	11
12	7	1
13	7	0
14	14	1
15	10	1
16	15	11
17	7	14
18	7	24
19	7	25

<i>Point</i>	X	Y
20	14	24
21	10	24
22	15	14
23	1	11
24	0	7
25	3	7
26	2	11
27	1	14
28	0	18
29	3	18
30	2	14
31	0.5	12.5
32	3	12.5
33	19	11
34	22	11
35	19	12.5
36	22	12.5
37	19	14
38	22	14

<i>Ligne</i>	<i>Origine</i>	<i>Destination</i>	<i>Ligne</i>	<i>Origine</i>	<i>Destination</i>
1	1	2	16	19	20
2	2	3	17	20	18
3	3	4	18	21	22
4	4	5	19	23	24
5	5	6	20	24	25
6	6	7	21	25	26
7	6	8	22	27	28
8	8	9	23	28	29
9	9	10	24	29	30
10	10	1	25	31	32
11	11	13	26	33	37
12	13	14	27	37	38
13	14	12	28	38	34
14	15	16	29	34	33
15	17	19	30	35	36

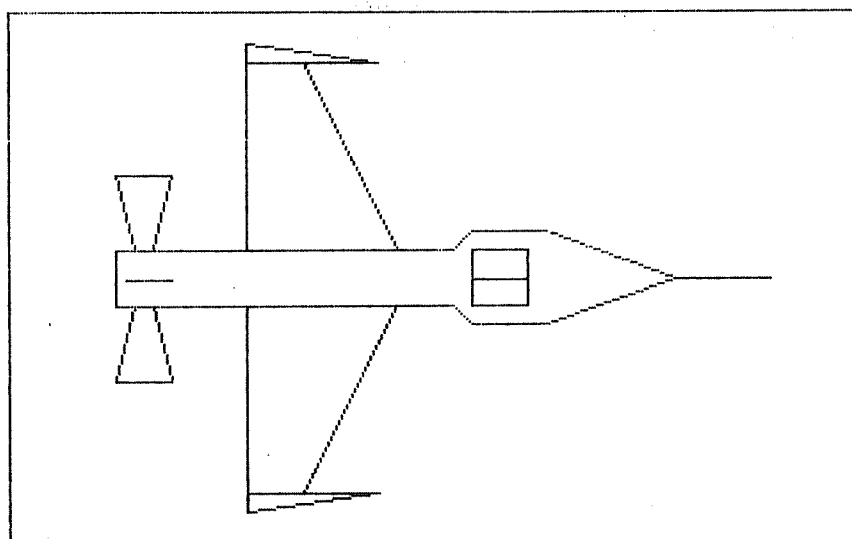


Fig. 2.19

Le LRP est fixé au dernier point calculé. Dans l'exemple précédent le LRP est fixé à (319,100). Les boîtes (rectangles pleins) ont le même problème que PSET et PRESET entre les points 320 et 639 : bien que les coordonnées soient en dehors de l'écran, BASIC insiste pour les tracer. Par exemple la boîte LINE (300,100)-(500,120),3,BF, est séparée en

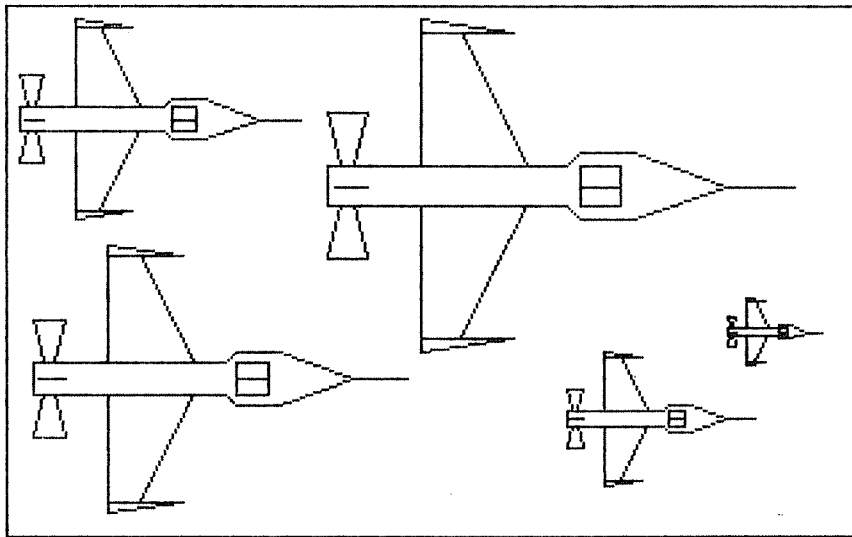


Fig. 2.20

deux : une équivalente à la boîte produite par l'instruction LINE (300, 100)–(319,120),3,BF et une seconde équivalente à celle produite par LINE (0,102)–(180,122),3:BF (ou LINE (0,102)–(500–320,122),3, BF).

REVISION

- | | |
|-----------------------------|---|
| LINE(10,10)–(50,50) | Dessine une droite entre les points (10,10) et (50,50) et laisse le LRP à (50,50). |
| LINE(10,15)–STEP(50,40) | Dessine une droite du point (10,15) au point (60,55) en ajoutant 50 à 100 et 40 à 15. Le LRP est (60,55). |
| LINE STEP(10,15)–(20,40) | Dessine une droite à partir du point situé 10 pixels sur la droite et 15 en bas du LRP jusqu'au point (20,40). Le LRP devient (20,40). |
| LINE STEP(10,20)–STEP(7,15) | Dessine une droite à partir du point situé 10 pixels à droite et 20 pixels en bas du LRP jusqu'au point situé 7 pixels à droite et 15 pixels en haut du premier point, et fixe le LRP à ce dernier point. |

LINE-(10,50)	Dessine une droite à partir du LRP jusqu'au point (10,50). Le LRP devient (10,50).
LINE-STEP(-5,2)	Dessine une droite entre le LRP et le point situé 5 pixels sur la gauche et 2 en bas. Le LRP devient ce dernier point.
LINE(5,10)-(40,50),2	Dessine une droite du point (5,10) au point (40,50) avec la couleur 2. Le LRP est (40,50).
LINE(5,10)-(40,50),2,B	Dessine le cadre déterminé par les points (5,10) et (40,50) avec la couleur 2, laissant le LRP à (40,50). Ce cadre est équivalent à la suite d'instuctions suivantes : LINE(5,10)-(40,10),2: LINE(40,10)-(40,50),2: LINE(40,50)-(5,50),2: LINE(-5,50)-(5,10),2:
LINE(5,10)-(40,50),1,BF	Dessine le cadre déterminé par les points (5,10) et (40,50) comme on l'a expliqué pour l'instruction précédente, mais maintenant avec la couleur 1. Le LRP est (40,50).

EXERCICES

1. Ecrire un programme qui réalise la figure 2.21.
2. Dans le programme 2.11 nous avons étudié la symétrie d'un cercle par rapport à l'axe des X. La figure 2.22 nous montre qu'un cercle est également symétrique par rapport à l'axe des Y. Ecrire un programme qui dessine un cercle complet en calculant seulement la valeur de 0 au rayon.
3. La figure 2.23 montre que le cercle est symétrique par rapport aux lignes qui se trouvent à 45 degrés des axes. Ecrire un programme qui dessine un cercle en calculant seulement les valeurs de 0 à $\text{rayon} \times 0,71$.
4. Ecrire un programme d'écrtage utilisant l'équation de la droite déterminée par ses deux points extrêmes. Une fois l'équation calculée, le premier point situé à l'intérieur de l'écran peut être trouvé par une simple opération. Toutefois si l'intersection est négative, le premier point autorisé (s'il y en a) doit être trouvé avec la même équation mais en prenant Y comme variable dépendante.

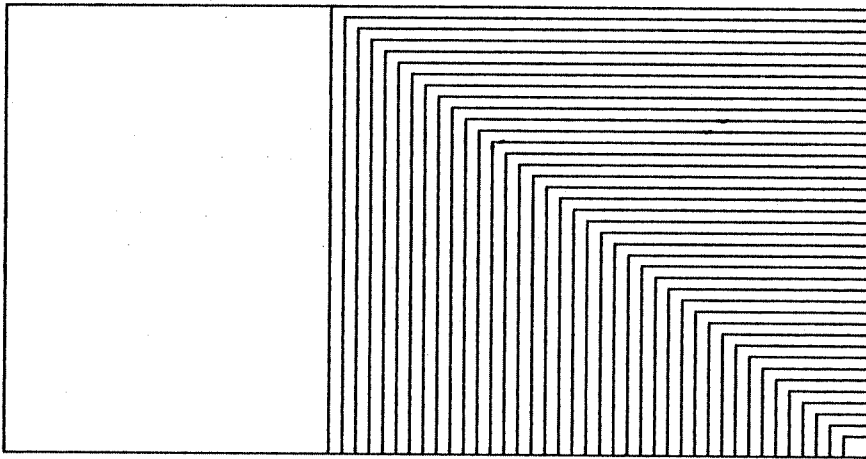


Fig. 2.21

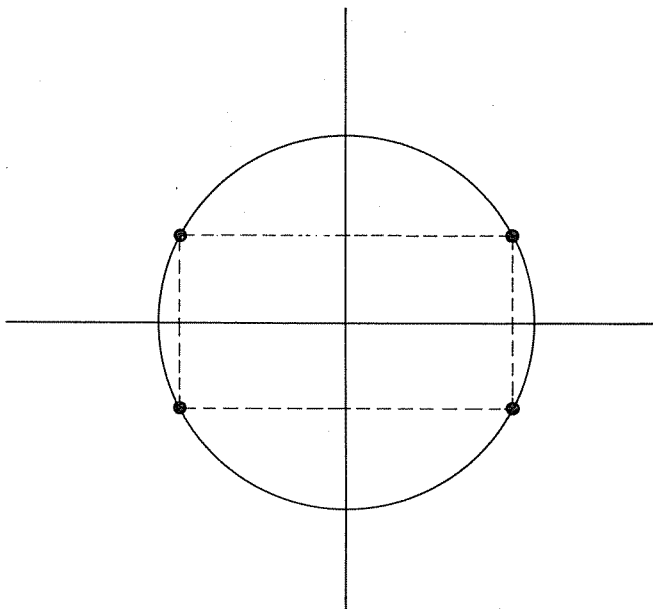


Fig. 2.22

NOTES

1. On a tout à fait le droit de spécifier un point de départ situé à droite du point d'arrivée, toutefois nous utiliserons toujours par convention des points de départ situés à gauche du point d'arrivée.

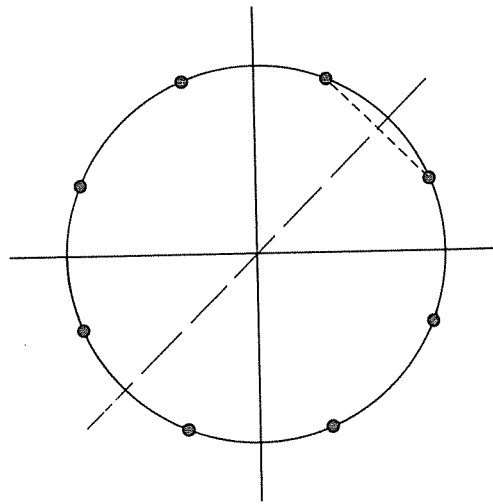


Fig. 2.23

2. Dans ce cas on peut utiliser le terme fonction ou équation. Cependant une fonction n'est pas obligatoirement identique à une équation.
3. A cause de la troncature, $f(x)$ peut être différent de X^2 ; cependant la différence sera très petite, en général insignifiante.
4. Le cercle a plus de symétries, comme on l'explique aux exercices 2.2 et 2.3.

3

Formes circulaires

On trouve fréquemment des formes circulaires dans les graphiques, par exemple : la lune, le soleil, les yeux, les roues, un arc en ciel... !

Au chapitre 2 nous avons étudié une manière de dessiner des cercles à l'aide de l'instruction LINE. Comme les cercles sont des formes très importantes, BASIC a une instruction pour les dessiner, à une vitesse qui ne peut être atteinte par les méthodes que nous avons étudiées jusqu'ici.

En géométrie le terme « cercle » désigne la zone à l'intérieur d'une circonférence, et le terme « circonférence » désigne la courbe qui entoure le cercle. L'instruction CIRCLE dessine en fait la circonférence, pas le cercle, mais nous utiliserons le terme cercle pour désigner la ligne, pas la zone à l'intérieur de celle-ci.

3.1 LES CERCLES

Le format de l'instruction est CIRCLE (*colonne,ligne*), *rayon*, *couleur*. Les coordonnées *colonne* et *ligne* déterminent le centre du cercle, et peuvent être données sous la forme absolue ou relative. Si *couleur* est omise, la couleur 3 est prise par défaut. Après l'exécution de l'instruction CIRCLE, le LRP est le centre.

Par exemple, l'instruction CIRCLE(100,20),30,1 dessine un cercle centré au point (100,20) avec un rayon de 30 et la couleur 1. L'instruction PSET(120,30):CIRCLE STEP(-15,10),50,2 tracera le point (120,30) et un cercle centré au point (105,40) parce que le centre donné sous forme relative, avec un rayon de 50 et la couleur 2.

Le programme 3.1 montre une utilisation intéressante des cercles. La boucle FOR des lignes 20 à 50 dessine des cercles concentriques (avec un centre commun) en incrémentant la valeur du rayon. La ligne 40 utilise C comme compteur. Tous les 7 cercles, la couleur est changée. Le graphique résultant est la cible de la figure 3.1. Pour changer la taille des anneaux de couleur, il faut changer le nombre qui est comparé avec C à la ligne 40.

```
0  "" 3- : Bull's eye ""
10 SCREEN 1:COLOR 0,1:CLS:COL=1
20 FOR R=1 TO 120
30  CIRCLE(160,100),R,COL
40  C=C+1:
    IF C > 7
      THEN
        COL=COL+1:C=0:
        IF COL=4
          THEN
            COL=0
50 NEXT
```

PROGRAMME 3.1

Si les cercles dessinés par le programme 3.1 ne sont pas concentriques et que le centre est déplacé à chaque itération de la boucle FOR, alors la forme résultante ressemblera à un long tube, l'inclinaison dépendant du facteur utilisé pour déplacer le centre. Le programme 3.2 demande, à la ligne 20, l'inclinaison désirée. Si la réponse est 1 par exemple, l'inclinaison sera 1, c'est-à-dire que le centre de chaque cercle sera décalé d'un pixel vers le bas chaque fois. Si la réponse est 2 chaque cercle aura son centre situé 2 pixels en dessous du centre du cercle précédent. Quand l'inclinaison est zéro, le centre du cylindre est au niveau de l'œil, exactement comme dans « la cible » du programme 3.1. Il y a deux cas intéressants : celui pour lequel l'Inclinaison = 0,84 et le Départ = 10 et celui pour lequel Inclinaison = 0,84, Départ = 190. La figure 3.2 nous montre le tube produit quand SP = 2 et IN = 1.1.

```
0  "" 3-2 : Cylinder ""
10 SCREEN 1:COLOR 0,1:COL=1:CLS
20 INPUT "Inclination ";IN
30 INPUT "Starting point ";SP:CLS
40 FOR R=1 TO 105
50  CIRCLE(160,SP+R*IN),R,COL
60  C=C+1:
    IF C > 5
      THEN
```

```

COL=COL+1;C=0;
IF COL=4
  THEN
    COL=0
70 NEXT

```

PROGRAMME 3.2

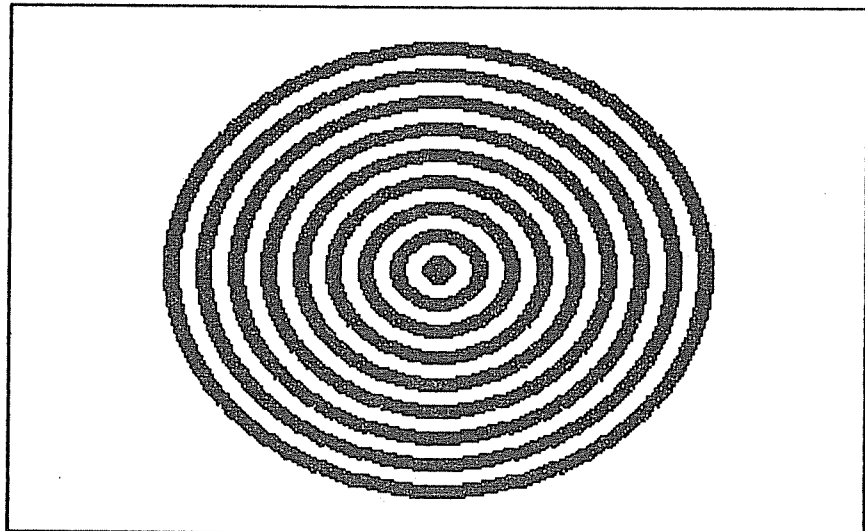


Fig. 3.1

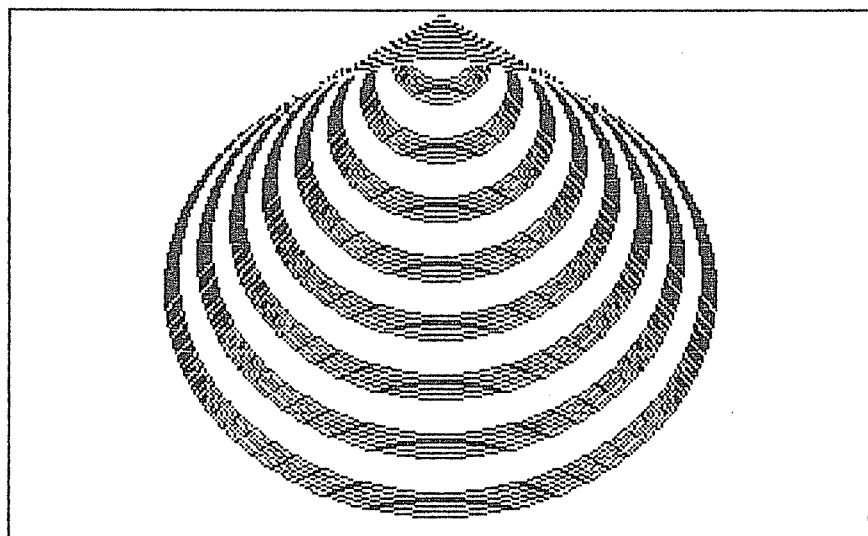


Fig. 3.2

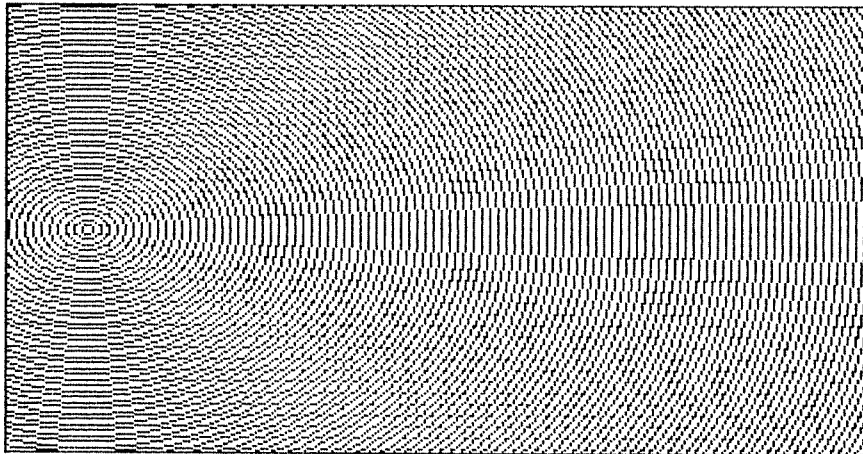


Fig. 3.3

3.1.1. Les cercles plus grands que l'écran

Quand le centre du cercle est proche de la marge de l'écran, ou quand le rayon est très grand, la totalité du cercle ne rentre pas dans l'écran. Dans ce cas la partie qui rentre est dessinée et le reste ignoré. Le programme 3.3 remplit l'écran avec des cercles concentriques dont certains sont trop grands pour être dessinés entièrement. Le résultat apparaît à la figure 3.3.

```
0  °° 3-3 : Fill with circles °°
10 SCREEN 1:CLS
20 FOR RADIUS=2 TO 312 STEP 3
30   CIRCLE (30,100),RADIUS
40 NEXT
```

PROGRAMME 3.3

Le programme 3.4 crée un cadre fantaisiste sur les quatre côtés de l'écran. Les centres ont été choisis de telle sorte que les cercles ne rentrent pas entièrement dans l'écran, et le résultat est que les coins sont colorés avec la couleur définie en ligne 10. Les lignes 100 et 110 dessinent des boîtes avec la couleur 0 pour effacer la partie intérieure des cercles, et la ligne 130 dessine un rectangle sur les bords de l'écran pour compléter le cadre. On peut voir le cadre résultant à la figure 3.4.

```
0  °° 3-4 : Fancy frame °°
10 INPUT "Color (1-3) ";C
20 SCREEN 1:COLOR 0,0:CLS
25 ' Draw the four circles
30 FOR R=50 TO 69
```

```

40  CIRCLE(49,41),R,C
50  CIRCLE(270,41),R,C
60  CIRCLE(49,158),R,C
70  CIRCLE(270,158),R,C
80  NEXT
90  ' Erase unwanted parts of the circles
100 LINE(0,47)-(319,160),0,BF
110 LINE(47,0)-(272,199),0,BF
120 ' Draw the rectangular frame
130 LINE(0,0)-(319,199),C,B

```

PROGRAMME 3.4

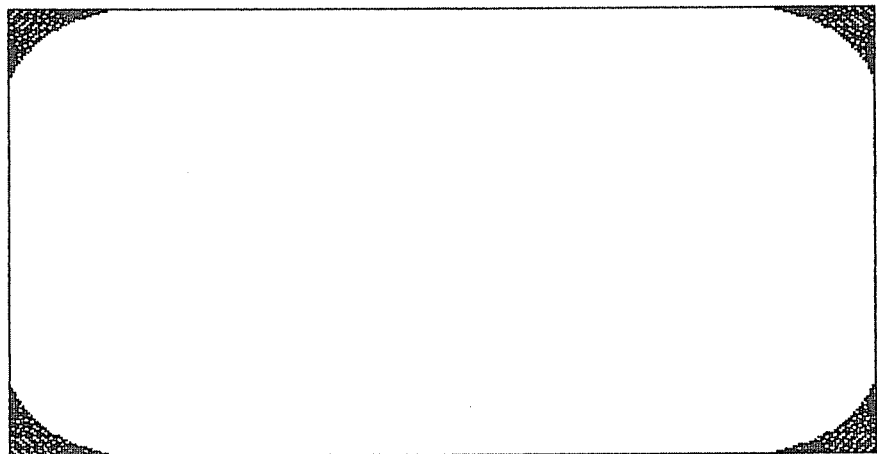


Fig. 3.4

3.1.2 Cercles centrés à l'extérieur de l'écran

On a tout à fait le droit de dessiner un cercle dont le centre ne soit pas à l'intérieur de l'écran. Si certains des points de ce cercle tombent dans la gamme de coordonnées valides, cette partie sera dessinée, sinon le cercle entier sera ignoré. Le programme 3.5 dessine des cercles concentriques centrés au point 500,500, manifestement hors de l'écran. La figure 3.5 nous montre le graphique résultant.

```

0  ' 3-5 : Center outside the screen °
10 SCREEN 1:CLS
20 FOR RADIUS=410 TO 770 STEP 10
30  CIRCLE (500,500),RADIUS:
40 NEXT

```

PROGRAMME 3.5

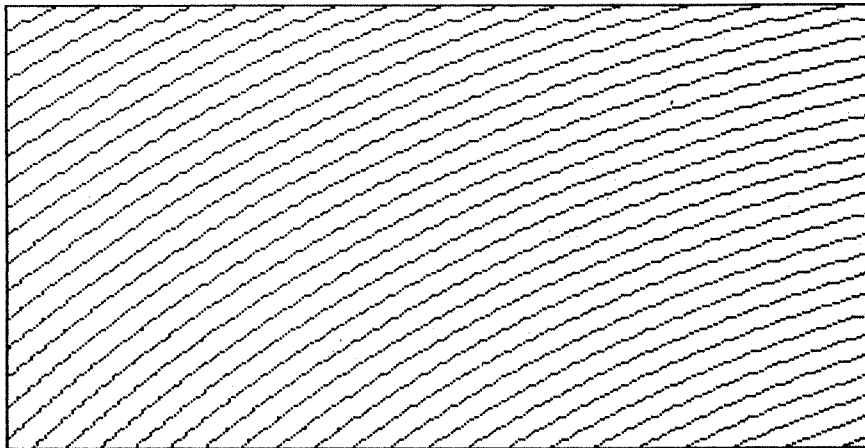


Fig. 3.5

3.2 LES ARCS

Au paragraphe précédent nous avons vu que lorsqu'une partie seulement du cercle rentre dans l'écran, un arc de cercle est dessiné. Il existe une seconde manière d'utiliser l'instruction CIRCLE, cette fois pour dessiner des arcs de cercle. Le format est CIRCLE (*colonne,ligne*),*rayon*,*couleur,début,fin*. *Début* et *fin* sont des valeurs d'angles entre 0 et $2 \times \pi$. L'angle 0 est sur une ligne horizontale sur la droite du centre de l'écran, et les valeurs positives croissent dans le sens contraire des aiguilles d'une montre. Ces valeurs sont en radians. Pour utiliser les degrés, multipliez le nombre par 0,01745329, c'est-à-dire $\pi/180$. Pour convertir les radians en degrés, multipliez par 57,29578, c'est-à-dire $180/\pi$. La figure 3.6 montre la correspondance entre les angles et la position de l'écran.

Si on donne *Début*, on doit aussi donner *Fin*. Le programme 3.6 dessine le sourire de la figure 3.7 à l'aide d'un demi-cercle et d'une partie d'un cercle plus grand, tous les deux se terminant aux mêmes points.

```
0  °° 3-6 : Smile °°
10 SCREEN 1:CLS
20 CIRCLE(160,100),70,3,3.141593,6.283186
30 CIRCLE(160,40),100,3,3.94,5.48
```

PROGRAMME 3.6

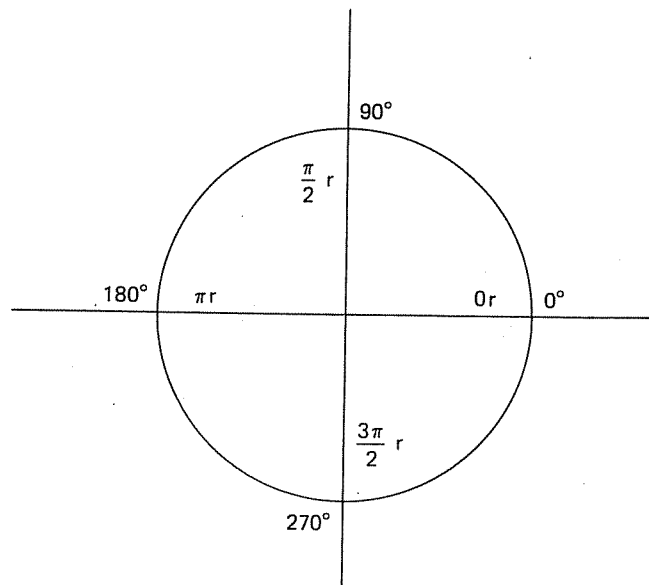


Fig. 3.6

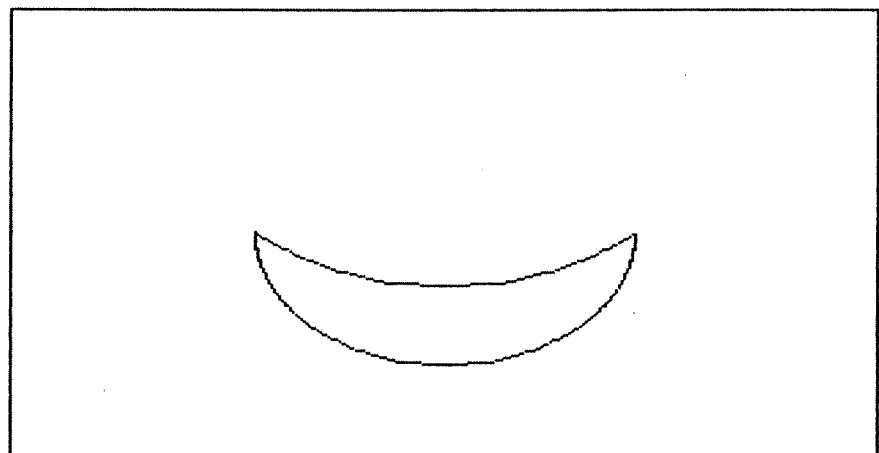


Fig. 3.7

Le programme 3.7 dessine un arc et un rectangle pour créer la forme de l'ampoule électrique rudimentaire qu'on voit à la figure 3.8.

```
0  *** 3-7 : Light bulb **
10 SCREEN 1:COLOR 0,1:CLS
20 CIRCLE(160,100),60,2,5.45,4
30 LINE(120,138)-(200,180),2,B
```

PROGRAMME 3.7

3.2.1 Arcs du haut et du bas

Les arcs produits par le programme 3.6 sont une partie du bas de cercles, alors que celui du programme 3.7 est une section du haut du cercle. Nous pouvons voir ici que la même paire de valeurs qu'on prend pour *début* et *fin* peut produire deux arcs différents. Quand *début* est plus petit que *fin*, l'arc produit est celui qui commence à *début* ; quand *fin* est plus petit l'arc produit commence à *fin*. Le programme 3.8 dessine deux arcs, l'un étant le complément de l'autre, c'est-à-dire que la somme des deux forme un cercle complet. La ligne 30 trace l'angle commençant à l'angle 0 (la partie droite de l'axe des X) avec la couleur 2, et la ligne 40 dessine l'arc commençant là où l'autre finit et allant jusqu'au point où l'autre commence.

```
0  "" 3-8 : Complementary arcs ""
10 SCREEN 1:COLOR 0,1:CLS
20 FOR S=0 TO 6.28 STEP .1
30  CIRCLE(160,100),80,2,0,S
40  CIRCLE(160,100),80,1,S,0
50 NEXT
```

PROGRAMME 3.8

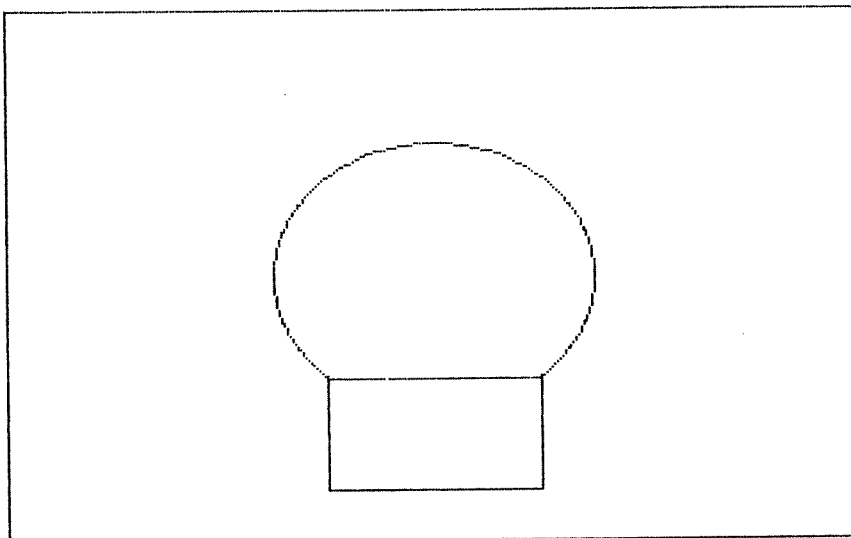


Fig. 3.8

3.2.2 Les hélices

Nous avons maintenant tous les outils nécessaires pour dessiner une hélice. La figure 3.9 montre l'étape initiale de cette procédure : il y a deux centres, A et B , séparés par une distance R .

Dans une première étape, la moitié du cercle est dessinée avec A comme centre et R comme rayon. Un second demi-cercle est alors tracé avec B comme centre et $2 \times R$ (le rayon précédent plus R) comme rayon. Le décalage du centre de R unités est compensé par l'accroissement du rayon, et les deux demi-cercles se rencontrent. Dans une troisième étape, le demi-cercle est centré en A et le rayon est à nouveau incrémenté de R unités, faisant se rencontrer les deux demi-cercles extérieurs. En répétant cette procédure, on dessine l'hélice de la figure 3.10.

La ligne 20 du programme 3.9 demande l'incrément désiré pour le rayon et trace l'hélice selon la méthode décrite ci-dessus.

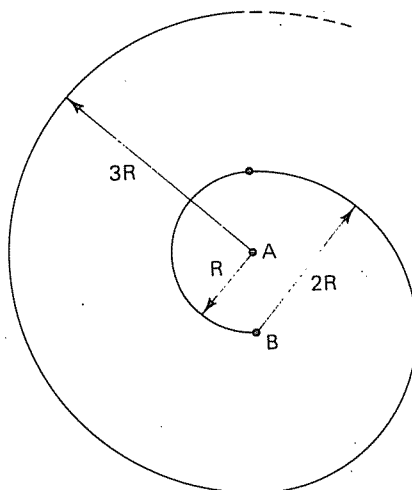


Fig. 3.9

```
0  ** 3-9 : Spiral **
10 SCREEN 1:CLS
20 INPUT"width ";W
30 SCREEN 1:COLOR 0,0:CLS
40 LINE(0,0)-(319,199),3,B
50 R=W:Y=100
60 CIRCLE(160,Y),R,,1.570797,4.71239,1
70 R=R+W:Y=Y+W
80 CIRCLE(160,Y),R,,4.71239,1.570797,1
90 R=R+W:Y=Y-W
100 IF R<=100
    THEN
        GO
```

PROGRAMME 3.9

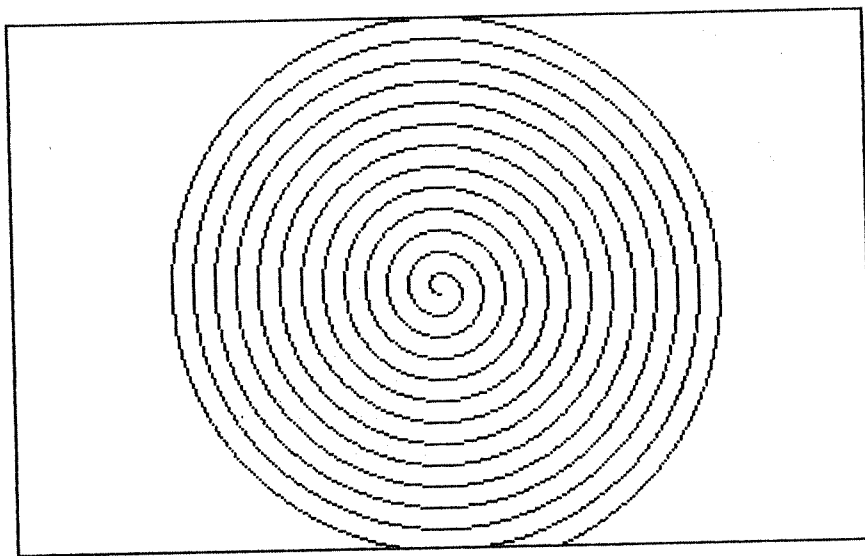


Fig. 3.10

Dans le programme 3.31 nous verrons une manière plus précise pour tracer les hélices.

3.2.3 La connexion des arcs avec le centre

Si les valeurs prises pour *début* et/ou *fin* sont négatives, les points finissant l'arc sont connectés avec le centre et c'est la valeur absolue des arcs qui est prise pour *début* et *fin*. L'instruction `CIRCLE(160,150),130,2,-1,-2`, par exemple, dessine un camembert. Le programme 3.10 dessine une roue : à la ligne 10, F est le facteur de conversion des degrés en radians. Dans la boucle FOR, on prend S pour la fin de l'arc et PS pour son début. A la ligne 10 PS égale zéro, ainsi au premier passage dans la boucle, l'arc est dessiné de 0 à 10 degrés. Quand l'arc est dessiné à la ligne 40, PS et S sont multipliés par F pour travailler sur des valeurs exprimées en radians. Voyez la roue obtenue à la figure 3.11.

```
0  ** 3-10 : Wheel **
10 F=1.745329E-02:PS=0
20 SCREEN 1:COLOR 0,0:CLS
30 FOR S=10 TO -360 STEP 10
40   CIRCLE(160,100),80,2,-PS*F,-S*F
50 NEXT
```

PROGRAMME 3.10

3.2.4 Les camemberts

Une application immédiate des arcs de circonférences connectés avec le centre est la création de camemberts. Sachant que le cercle complet a 2π radians, il est facile de calculer les valeurs à partir des différents morceaux du camembert. Supposons que T soit le total (la somme de tous les articles) et X soit le pourcentage² d'une section particulière, on peut calculer l'angle correspondant à cette valeur par la formule $ANGLE = 2\pi \cdot X / 100$. Par exemple les trois valeurs en pourcentage 10,20,70 produisent les angles suivants :

$ANGLE1 = 2 \cdot \pi \cdot 10 / 100$	($ANGLE1 = 0.6283186$)
$ANGLE2 = 2 \cdot \pi \cdot 20 / 100$	($ANGLE2 = 1.256637$)
$ANGLE3 = 2 \cdot \pi \cdot 70 / 100$	($ANGLE3 = 4.39823$)

En prenant 100 comme rayon et ces trois angles, les instructions pour créer un camembert ayant son centre au milieu de l'écran en moyenne résolution seront :

```
100 CIRCLE (160,100),100,1,-.001,-.6283186
110 CIRCLE (160,100),100,2,-.6283186,-1.884956
120 CIRCLE (160,100),100,3,-1.884956,-.001
```

Le programme 3.11 généralise cette idée et crée un camembert avec n'importe quel nombre de parts.

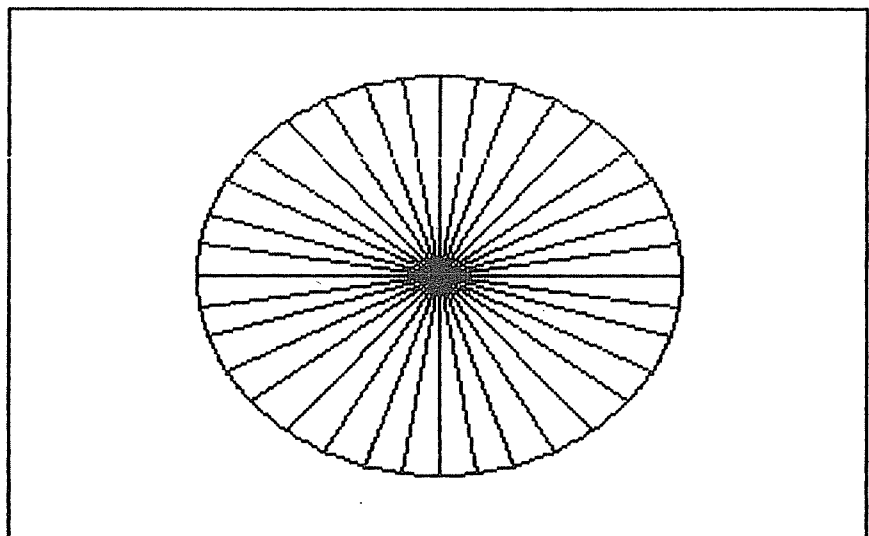


Fig. 3.11

```

0  ' ** 3-11 : Pie chart **
20 SCREEN 1:CLS
30 DEF FN ANG(X)=2*PI*X/100
40 PI=3.141593:A=0
50 INPUT"Number of items ";N
60 DIM M(N)
70 FOR I=1 TO N:
    PRINT"Item # ";I;:INPUT M(I):
    NEXT:CLS
80 FOR I=1 TO N:
    T=T+M(I):
    NEXT
90 FOR I=1 TO N:
    M(I)=M(I)*100/T:
    NEXT
100 FOR I=1 TO N
110  V=FN ANG(M(I)):T=A+V:
    IF T>6.283186
        THEN
            T=0
120 CIRCLE(159,99),119,1 + I MOD 3,-A,-(T):A=T
130 NEXT

```

PROGRAMME 3.11

Aux paragraphes 4.6, 7.19 et 12.7 nous expliquerons d'autres concepts relatifs aux camemberts.

3.3 LES ELLIPSES

Les ellipses ne sont que des cercles allongés, et les cercles sont des cas particuliers d'ellipses. Si ce n'est pas précisé par ailleurs, l'instruction CIRCLE trace par défaut un cercle régulier, mais cette instruction a une troisième syntaxe qui permet la création d'ellipses orientées horizontalement ou verticalement. La manière d'indiquer à l'instruction CIRCLE que nous voulons une ellipse et non un cercle est de spécifier le rapport entre l'axe des X et l'axe des Y. Le format est CIRCLE (*colonne,ligne*), *rayon,couleur,début,fin,aspect* où *couleur* et/ou *début,fin* peuvent être omis (bien qu'il faille toujours mettre les virgules). Quand *aspect* n'est pas précisé, c'est la valeur 5/6 qui est prise par défaut. Ceci signifie que si par exemple le rayon donné est 100, le rayon sur l'axe des X sera 100 (quand *aspect* est inférieur à 1, *rayon* devient le rayon sur l'axe des X) et il sera de 83($100 \cdot 5/6 = 83,33$) sur l'axe des Y. Ceci pour compenser la différence de taille horizontale et verticale des pixels³. Si une valeur est précisée pour *aspect*, les rayons correspondants seront pris pour les axes X et Y.

L'instruction CIRCLE trace toujours des ellipses qui seront parallèles aux axes horizontal ou vertical. Il est par exemple impossible de l'utiliser pour tracer une ellipse ayant son plus grand rayon incliné de 45 degrés par rapport à l'un des axes. Comme nous l'avons déjà mentionné, si

aspect est inférieur à l'unité, le rayon donné est pris pour rayon suivant l'axe de X. La figure 3.12 montre plusieurs ellipses réalisées par le programme 3.12 et toutes de grands axes parallèles à l'axe des X.

```
0 ' ** 3-12 : Horizontal ellipses **
10 SCREEN 1:CLS
20 FOR I=.63 TO .01 STEP -.03
30   CIRCLE(159,100),157,2,,,I
40 NEXT
```

PROGRAMME 3.12

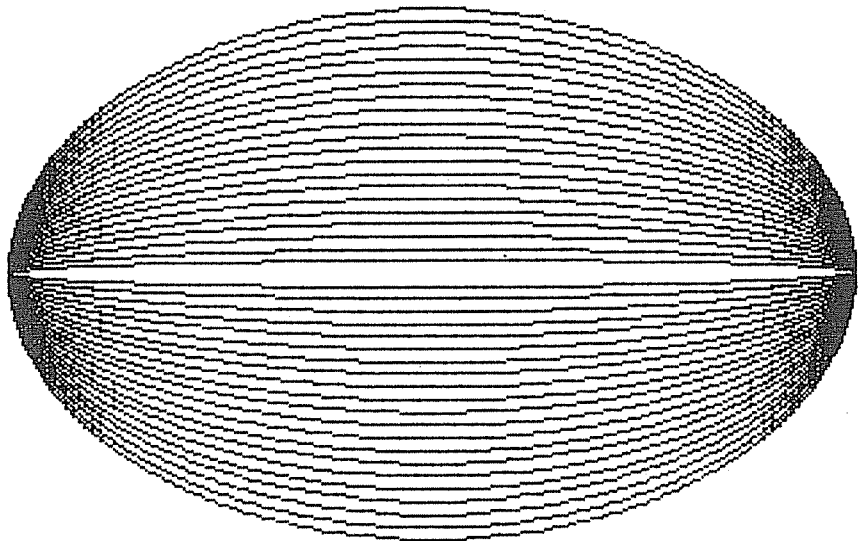


Fig. 3.12

Si *aspect* est supérieur à un, les ellipses sont parallèles à l'axe des Y et le rayon donné est pris pour rayon suivant l'axe des Y. La figure 3.13 montre les ellipses verticales réalisées par le programme 3.13.

```
0 ' ** 3-13 : Vertical ellipses **
10 SCREEN 1:COLOR 0,0:CLS
20 I=1.1
30 WHILE I<30
40   CIRCLE(159,100),95,1,,,I
50   I=I^1.3
60 WEND
```

PROGRAMME 3.13

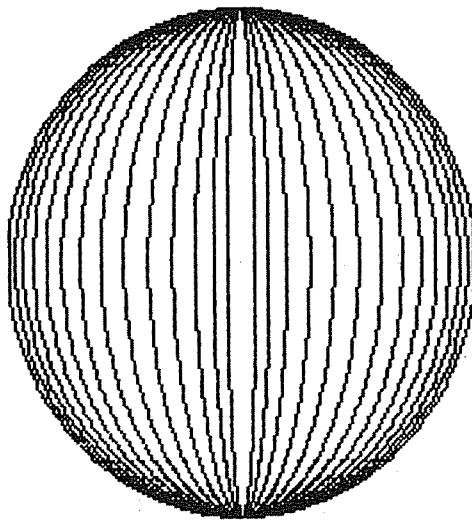


Fig. 3.13

Le programme 3.14 dessine un cercle du rayon précisé à la ligne 15 et dessine des ellipses horizontales et verticales pour remplir le cercle et créera la forme ressemblant à une planète que l'on voit à la figure 3.14.

```

0  ** 3-14 : Mixed ellipses **
10 SCREEN 1:COLOR 0,0
15 CLS:INPUT"Center of the world ";X,Y
20 IF X>319
    THEN
        X1=319
    ELSE
        X1=X
25 IF Y>199
    THEN
        Y1=199
    ELSE
        Y1=Y
30 S=5
40 CLS
50 CIRCLE(X,Y),X,3,,,1:PAINT(X1,Y1),3,3
60 FOR I=58 TO 1 STEP -1
70   CIRCLE(X,Y),X,C MOD 3,,,I/58
80   C=C+1
85 NEXT
90 I=2
100 WHILE I<102
110   CIRCLE(X,Y),X,2,,,100/I
120   I=I+S:S=S/1.04
125 WEND
130 W$=INPUT$(1):GOTO 15

```

PROGRAMME 3.14

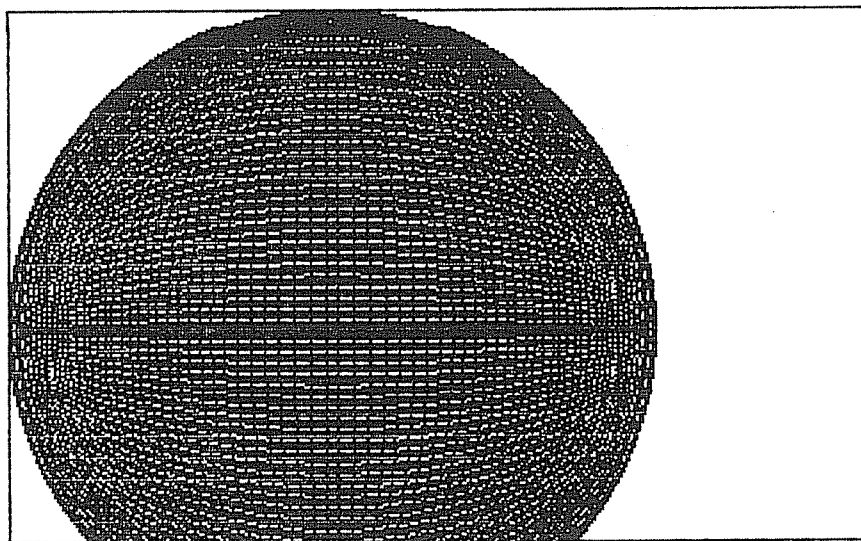


Fig. 3.14

3.3.1 Arcs d'ellipses

Si *début* et *fin* sont omis, comme dans l'instruction CIRCLE(160, 100),120,2,,,5, la totalité de l'ellipse sera dessinée, sinon ces deux paramètres indiquent les points de départ et de fin de l'ellipse, comme on l'a vu au paragraphe 3.2.1.

Le programme 3.15 dessine le ballon de rugby avec des bandes colorées de la figure 3.15. Chaque bande est en fait un ensemble d'arcs.

```
0  °° 3-15 : Football °°
10 SCREEN 1:CLS
20 FOR I=.6 TO 0 STEP -.004
30   FOR J=0 TO 6 STEP .7853982
40     CIRCLE(160,100),150,1+J MOD 3,J,J+.7853983,I
50   NEXT
60 NEXT
```

PROGRAMME 3.15

3.4 LES COORDONNEES POLAIRES

Le concept de coordonnées polaires est très lié aux formes circulaires. Le système de coordonnées que nous avons étudié dans les paragraphes

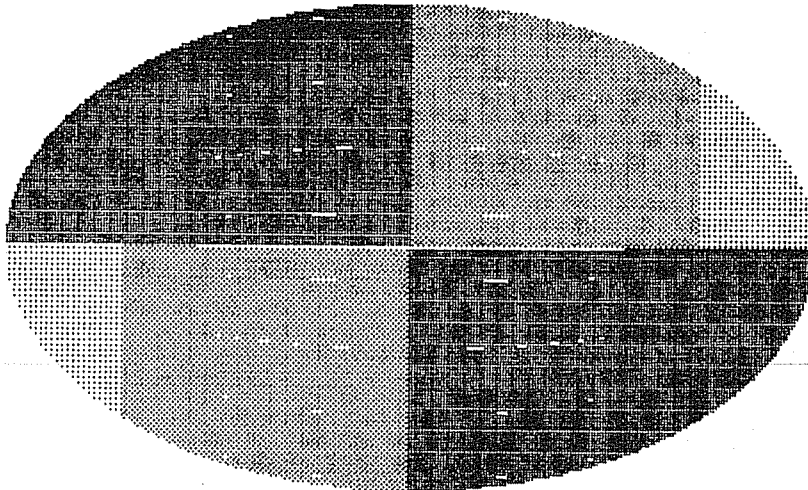


Fig. 3.15

précédents est appelé système de coordonnées cartésiennes. Dans ce dernier, chaque point dans le plan X-Y est déterminé d'une manière unique par un couple de nombres qui représentent sa position suivant l'axe des X et des Y. Les coordonnées polaires déterminent également chaque point au moyen de deux valeurs : un angle et une distance au centre du système (l'équivalent du point (0,0) des coordonnées cartésiennes). Le point dont les coordonnées cartésiennes sont (0,30) dans le plan X-Y peut être déterminé de manière unique en coordonnées polaires par la paire (30,90), c'est-à-dire une distance de 30 unités du centre (appelée habituellement le rayon) et un angle de 90 degrés par rapport à l'angle 0, ou l'axe angulaire, comme le montre la figure 3.16.

Le travail avec des lignes droites est plus compliqué dans ce système que dans le système cartésien, mais manipuler des cercles, des ellipses et des formes circulaires devient extrêmement simple. Par exemple, pour calculer chaque point d'un cercle, l'équation cartésienne que nous utilisions au paragraphe 2.5.4 utilise deux exponentiations et une racine carrée ; en coordonnées radiales l'équation devient simplement $R = C$ où C est le rayon. Cette équation signifie que le rayon est constant et n'est pas affecté par l'angle.

3.4.1 Le tracé d'équations polaires

Comme chaque point est adressé par un couple de nombres dans les coordonnées cartésiennes, tracer une fonction en équation polaire demande une étape supplémentaire : les deux équations :

$$X = \text{RAYON} \times \cos(\text{ANGLE}) \quad (3.1)$$

$$Y = \text{RAYON} \times \sin(\text{ANGLE}) \quad (3.2)$$

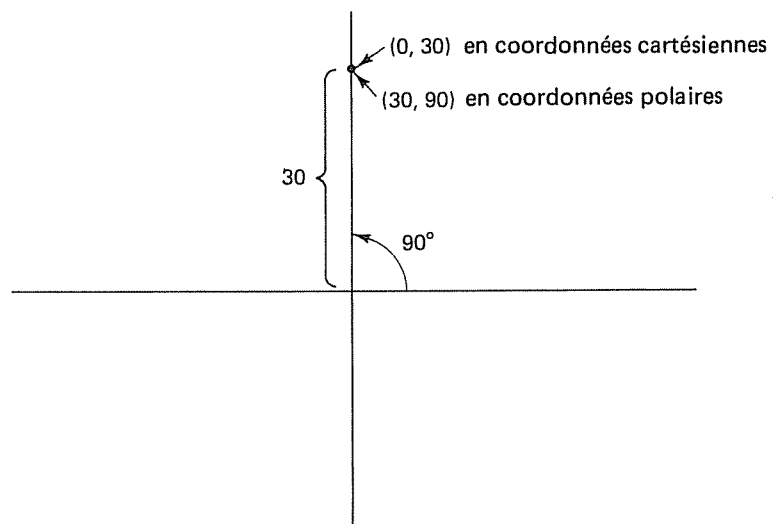


Fig. 3.16

déterminent les coordonnées cartésiennes de n'importe quel point donné dans un système de coordonnées polaires avec le centre au point (0,0). Le programme 3.16 trace un cercle en coordonnées polaires.

```

0  ° 3-16 : Circle, polar coordinates °
10 SCREEN 1:COLOR 0,0:CLS
20 RADIUS=80:F=1.745329E-02
30 FOR ANGLE=0 TO 360 STEP 5
40   X=RADIUS°COS(ANGLE°F)
50   Y=RADIUS°SIN(ANGLE°F)
60   PSET(160+X,100+Y)
70 NEXT

```

PROGRAMME 3.16

A la ligne 20, le rayon est fixé à 80 et F contient la constante pour convertir les degrés en radians. Les équations 3.1 et 3.2 font la conversion dans un système cartésien avec comme centre (0,0). Nous avons choisi comme centre du système le point (160,100), nous ajoutons donc ces valeurs à X et à Y à la ligne 60. Pour avoir un programme plus général, on devra utiliser deux variables pour les coordonnées cartésiennes du centre.

Dans la boucle FOR des lignes 30 à 70, ANGLE prend les valeurs de 0 à 360, par pas de 5. A cause de cela le cercle est dessiné comme une suite de points disjoints. Pour tracer une ligne continue, utilisez la méthode expliquée au paragraphe 2.5.

Comme l'angle va de 0 à 360 c'est toute la circonférence qui est tracée. Si la ligne 30 était changée par exemple en :

```
30 FOR ANGLE=0 TO 7000
```

les mêmes points seraient tracés plusieurs fois, l'un sur l'autre parce que toutes les valeurs angulaires sont traitées modulo 360 (un angle de $360 + X$ a le même sinus et cosinus que l'angle X).

3.4.2 Simulation d'instructions circulaires avec les coordonnées polaires

Nous avons déjà vu comment tracer des cercles avec les coordonnées polaires. Dans ce paragraphe nous expliquons comment simuler toutes les formes de l'instruction CIRCLE.

Arc. Pour dessiner une partie de cercle, changez simplement les valeurs limites de la boucle FOR qui contrôle l'angle. Le programme 3.17 demande les valeurs limites aux lignes 30 et 40. Comme les valeurs prises par ANGLE sont multipliées par F, les angles de début et de fin doivent être rentrés en degrés.

```
0  "" 3-17 : Arc, polar coordinates ""
10 SCREEN 1:COLOR 0,0:CLS
20 F=1.745329E-02:RADIUS=90
30 INPUT"Starting point (degrees) ";START
40 INPUT"Ending point (degrees) ";ENDING:
   IF ENDING < START
       THEN
           40
       ELSE
           CLS
50 FOR ANGLE=START TO ENDING STEP 2
60   X=RADIUS*COS(ANGLE*F)
70   Y=RADIUS*SIN(ANGLE*F)
80   IF ANGLE=START
       THEN
           PSET(160+X,100+Y)
       ELSE
           LINE-(160+X,100+Y)
90 NEXT
```

PROGRAMME 3.17

Si la valeur donnée pour ENDING est plus grande que START, le programme la rejettera. A la différence de l'instruction CIRCLE, les arcs

supérieurs produits par une paire d'angles limites doivent être signalés par un angle de départ négatif. Pour tracer un arc de $\text{ANGLE} = 270$ à $\text{ANGLE} = 180$ par exemple, les valeurs rentrées doivent être $\text{START} = -90$ (équivalent à 270 degrés) et $\text{ENDING} = 180$.

Connexion des limites des arcs avec le centre. Il est plus facile de connecter les extrémités des arcs avec le centre ici qu'avec l'instruction **CIRCLE** parce que le programme connaît à chaque instant les coordonnées des points utilisées. Le programme 3.18 dessine une roue. A la ligne 80, quand l'angle est un multiple de 20 (il y a dix-huit angles de 20 degrés dans 360 degrés) le point du cercle est connecté avec le centre.

```
0  °° 3-18 : Wheel, polar coordinates °°
10 SCREEN 1:COLOR 0,0:CLS
20 F=1.745329E-02:RADIUS=90
30 FOR ANGLE=0 TO 360
40   X=160+RADIUS*COS(ANGLE*F)
50   Y=100+RADIUS*SIN(ANGLE*F)
60   IF ANGLE>0
       THEN
           LINE(PX,PY)-(X,Y)
70   PX=X:PY=Y
80   IF ANGLE MOD 20=0
       THEN
           LINE(X,Y)-(160,100)
90 NEXT
```

PROGRAMME 3.18

L'intérêt de cette méthode pour tracer un cercle ou un arc, c'est qu'il est possible de connecter le centre avec des points situés sur la périphérie sans avoir en fait à tracer le cercle. La figure 3.17 montre les rayons d'une roue dessinée par le programme 3.19.

```
0  ° 3-19 : Connecting wires of wheel °
10 SCREEN 1:COLOR 0,0:CLS
20 F=1.745329E-02:RADIUS=90
30 FOR ANGLE=0 TO 360 STEP 10
40   X=160+RADIUS*COS(ANGLE*F)
50   Y=100+RADIUS*SIN(ANGLE*F)
60   LINE(X,Y)-(160,100)
70 NEXT
```

PROGRAMME 3.19

Ellipses. Les ellipses horizontales et verticales ne sont rien moins que des cercles dont les composantes de chaque point ont été affectées d'échelles

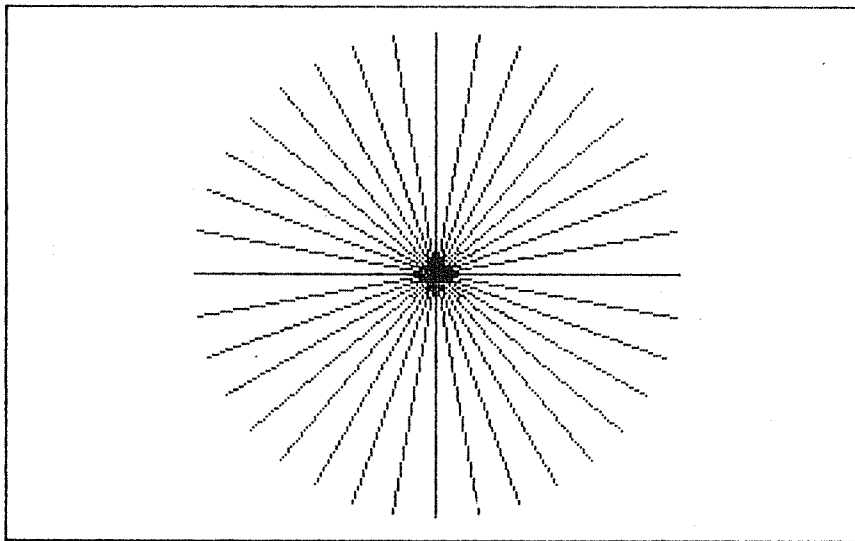


Fig. 3.17

différentes (On peut aussi dire que les cercles sont des cas particuliers d'ellipses.) Le programme 3.20 dessine une ellipse en multipliant les coordonnées X par 2, déformant ainsi le cercle horizontalement.

```
0  ** 3-20 : Ellipse, polar coordinates **
10 SCREEN 1:COLOR 0,0:CLS
20 FOR A=0 TO 6.29 STEP .1
30   X=2*80*COS(A)
40   Y=80*SIN(A)
50   PSET(X+160,Y+100)
60 NEXT
```

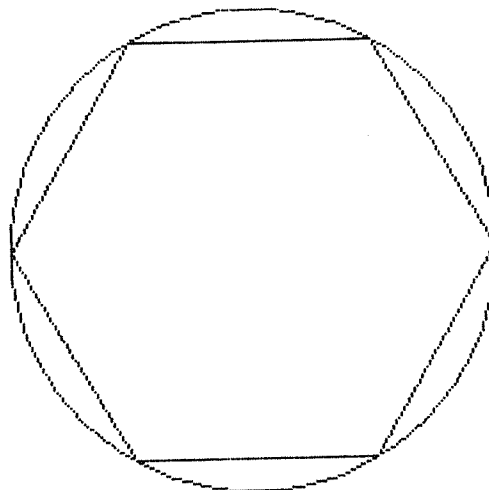
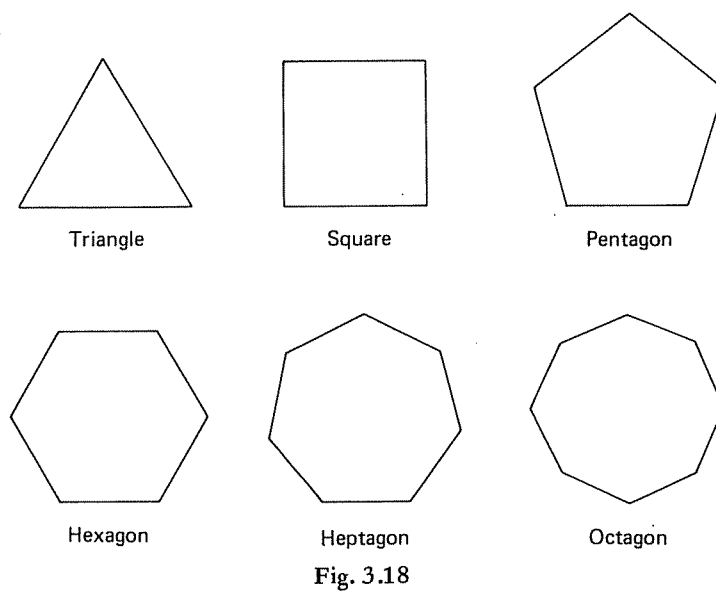
PROGRAMME 3.20

3.4.3 Les polygones

Les polygones réguliers sont des figures planes fermées ayant des côtés égaux (et par conséquent formant entre eux des angles égaux). Le polygone ayant le plus petit nombre de côtés est le triangle équilatéral. La figure 3.18 montre les six premiers polygones réguliers.

On peut voir facilement que tous les polygones réguliers peuvent être inscrits dans un cercle. La figure 3.19 montre comment tous les points externes d'un polygone se trouvent sur un cercle.

Un moyen pour dessiner facilement un polygone consistera à diviser la circonférence en autant de parties que le polygone a de côtés, calculer



les coordonnées de chaque point et les relier par des segments de droites. En fait tous les « cercles » tracés en coordonnées polaires dans ce chapitre sont assimilables à des polygones à un grand nombre de côtés.

Le programme 3.21 trace un polygone pouvant avoir n'importe quel nombre de côtés. Il faut donner le nombre de côtés, le rayon et le centre du cercle circonscrit. Ces valeurs sont entrées en lignes 30 à 50. Les figures 3.20 et 3.21 montrent les polygones créés par ce programme pour $S = 5$ et $S = 8$, le centre étant à (160,100) et le rayon 95.

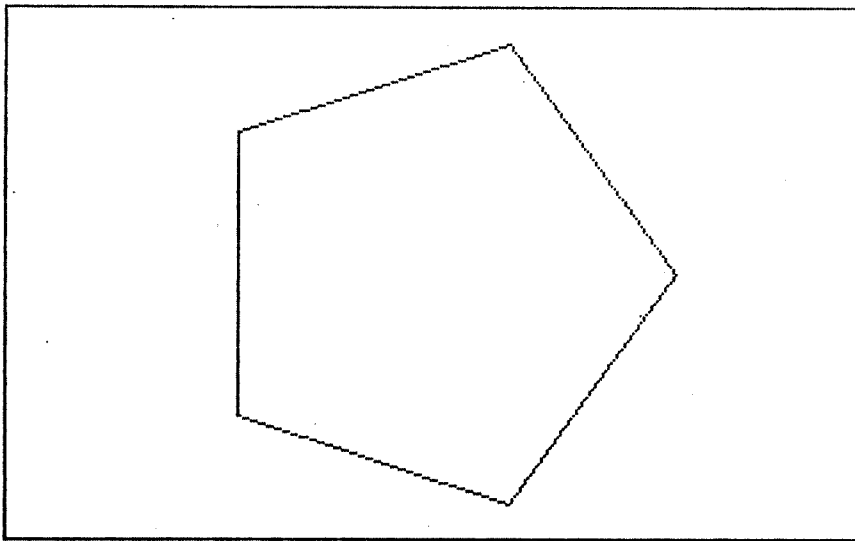


Fig. 3.20

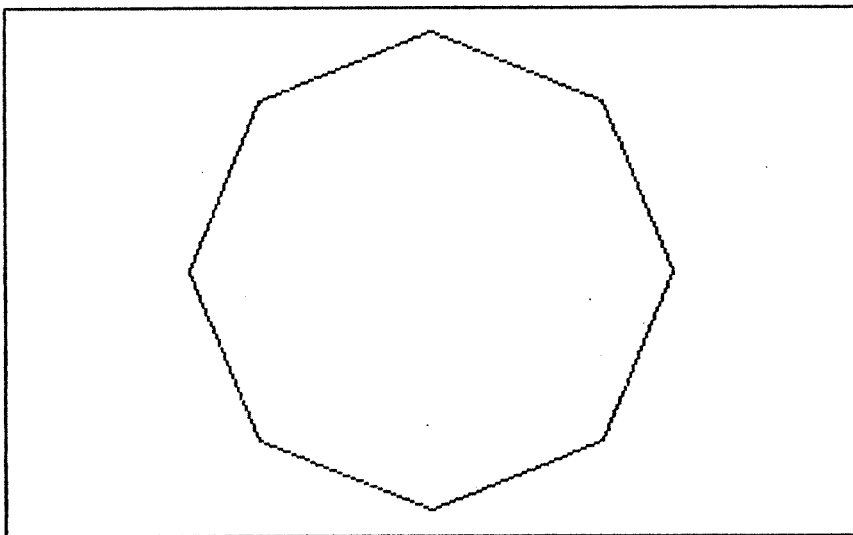


Fig. 3.21

```

O.  °° 3-21 : Polygon, polar coordinates °°
10 F=1.745329E-02
20 SCREEN 1:CLS
30 INPUT"Number of sides ";S
40 INPUT"Radius ";R

```

```

50 INPUT "Center (X,Y) ";XC,YC
60 CLS
70 FOR A=0 TO 360 STEP 360/S
80   X=XC+R°COS(A°F)
90   Y=YC+R°SIN(A°F)
100  IF A=0
      THEN
        PSET(X,Y)
      ELSE
        LINE-(X,Y)
110 NEXT

```

PROGRAMME 3.21

Pour relier les sommets du polygone au centre, changez simplement la ligne 100 du programme 3.21 et ajoutez la ligne 105 comme le montre le programme 3.22 qui suit :

```

100  IF A>0
      THEN
        LINE(PX,PY)-(X,Y)
105  LINE(X,Y)-(XC,YC):PX=X:PY=Y

```

PROGRAMME 3.22

La figure 3.22 montre un décagone avec ses sommets reliés au centre.

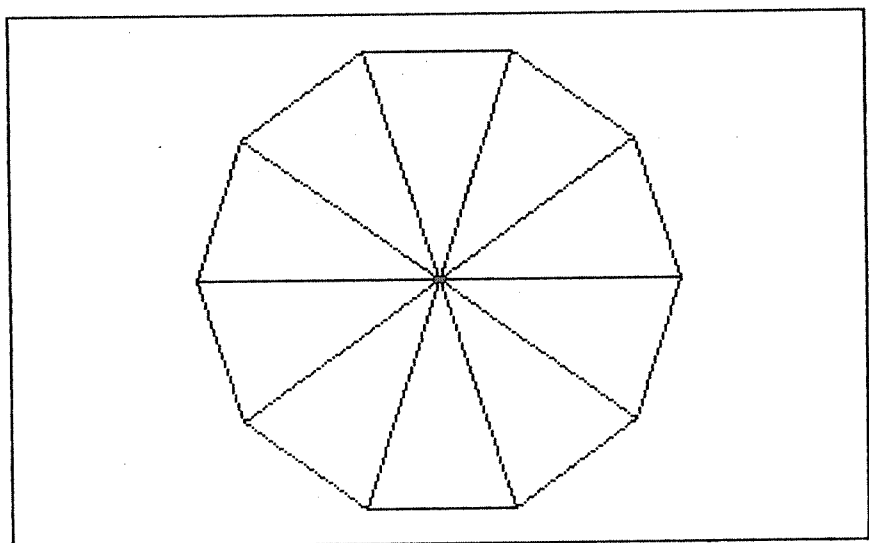


Fig. 3.22

3.4.4 Les étoiles

Si on relie tous les points d'un polygone, la forme résultante sera une étoile. Le programme 3.23 utilise cette méthode pour dessiner des étoiles pouvant avoir de 3 à 45 points.

```
0  ' ** 3-23 : Stars **
10 SCREEN 1:CLS
20 PI=3.141595:R=99
30 DIM X(55),Y(55)
40 FOR K=1 TO 16:
    READ W:F=360/W
50   FOR I=0 TO W-1:
        A=I*F
60     X=R*COS(A*PI/180):Y=R*SIN(A*PI/180)
70     X(I)=X+160:Y(I)=Y+100
80   NEXT
90   W$=INPUT$(1):CLS
100  FOR I=0 TO W-1
110    FOR J=I TO W-1
120      LINE(X(I),Y(I))-(X(J),Y(J))
130    NEXT
140  NEXT
150 NEXT
160 DATA 3,4,5,6,8,9,10,12,15,
        18,20,24,30,36,40,45
```

PROGRAMME 3.23

La figure 3.23 montre l'étoile à 12 côtés produite par le programme 3.23.

3.4.5 Equations en coordonnées polaires

Plusieurs équations en coordonnées polaires permettent de tracer des courbes complexes qui seraient difficiles à obtenir en coordonnées cartésiennes. Dans ce paragraphe nous montrerons certaines de ces équations sous la forme de petits programmes avec les figures correspondantes. En modifiant les paramètres des équations, on peut découvrir de nouveaux motifs.

```
0  '°° 3-24 : Core of Daisy °°
10 SCREEN 1:COLOR 0,1:CLS
20 T=80
30 FOR A=-30 TO 30 STEP 9.500001E-02
35   R=T*SIN((A+F)/3)
40   X=160+R*COS(A)
50   Y=100+R*SIN(A)
60   IF A=-30
```

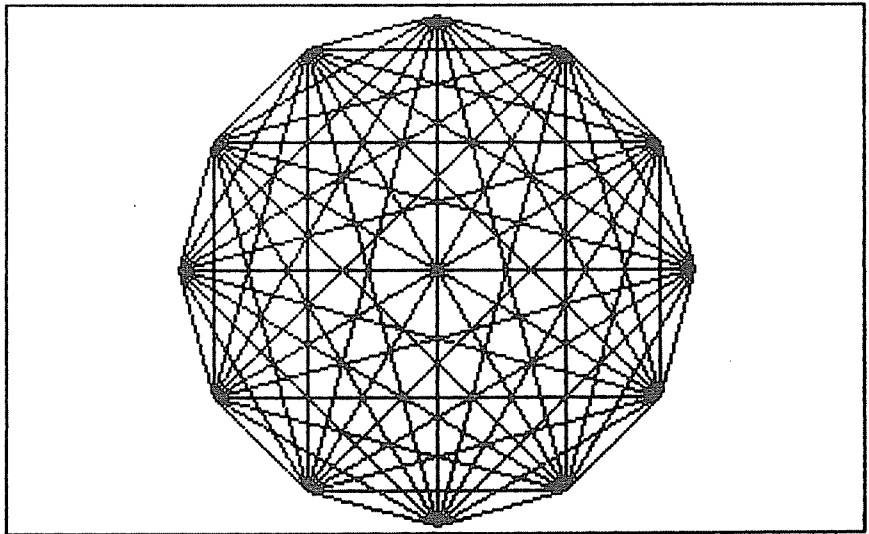


Fig. 3.23

```

      THEN
        PSET(X,Y)
      ELSE
        LINE-(X,Y)
65   F=F+.01
70  NEXT

```

PROGRAMME 3.24

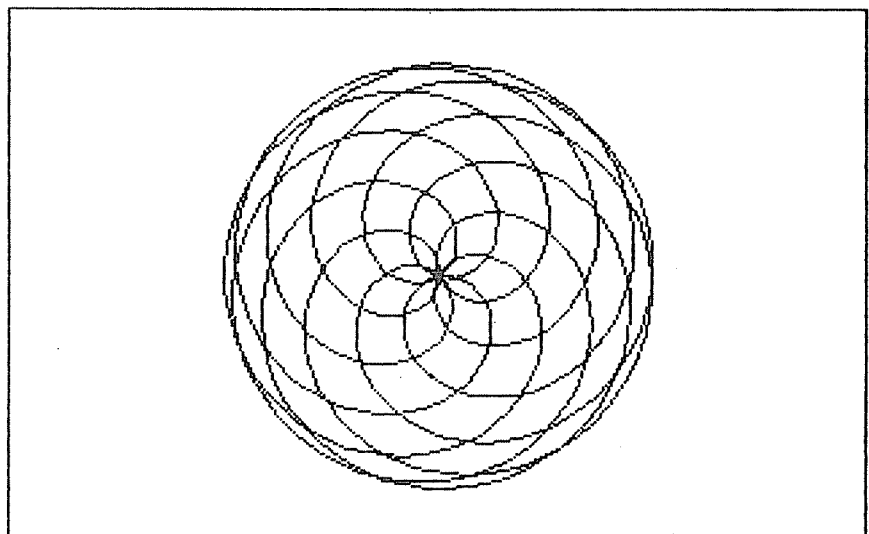


Fig. 3.24


```

0  °° 3-25 : Sea urchin °°
10 SCREEN 1:COLOR 0,1:CLS
20 T=80
30 FOR A=-40 TO 38.7 STEP .2
35   R=T°SIN((A+F)°.8)
40   X=160+R°COS(A)
50   Y=100+R°SIN(A)
60   IF A=-40
       THEN
           PSET(X,Y)
       ELSE
           LINE-(X,Y)
65   F=F+.01
70 NEXT

```

PROGRAMME 3.25

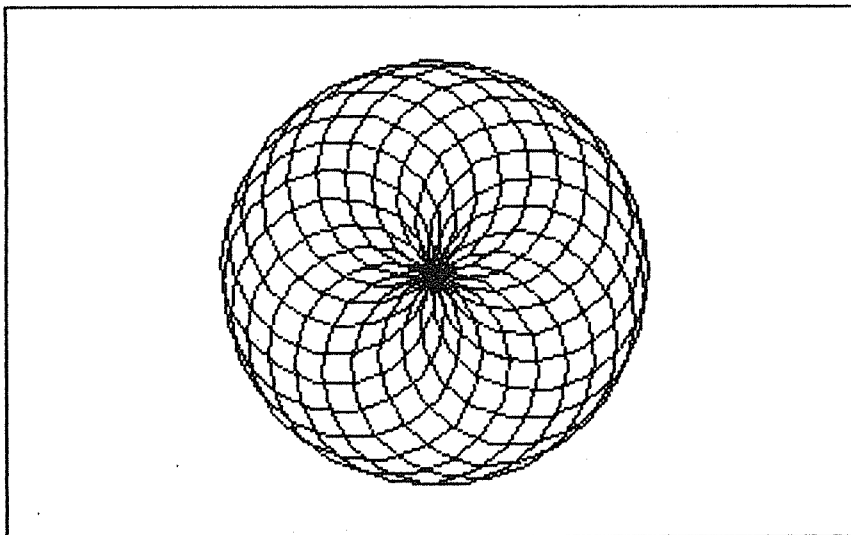


Fig. 3.25

```

0  °° 3-26 : Trinity °°
10 SCREEN 1:COLOR 0,1:CLS
20 T=80
30 FOR A=0 TO 22.1 STEP .0355
35   R=T°SIN((A+F)/3)
40   X=160+R°COS(A)
50   Y=100+R°SIN(A)
60   IF A=0

```

```

      THEN
        PSET(X,Y)
      ELSE
        LINE-(X,Y)
65   F=F+.01
70 NEXT

```

PROGRAMME 3.26

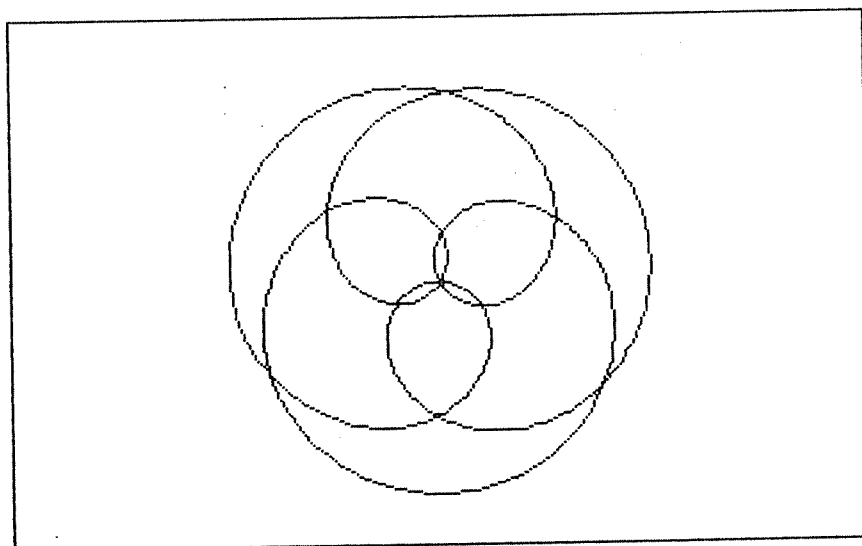


Fig. 3.26

```

0  '°° 3-27 : Circular spring °°
10 SCREEN 1:COLOR 0,1:CLS
20 T=95
30 FOR A=-30 TO 99 STEP .15
35   R=4+T°SIN((A+F)°.8)
40   X=160+R°COS(A)
50   Y=100+R°SIN(A)
60   IF A=-30
      THEN
        PSET(X,Y)
      ELSE
        LINE-(X,Y)
65   F=F+.01
70 NEXT

```

PROGRAMME 3.27

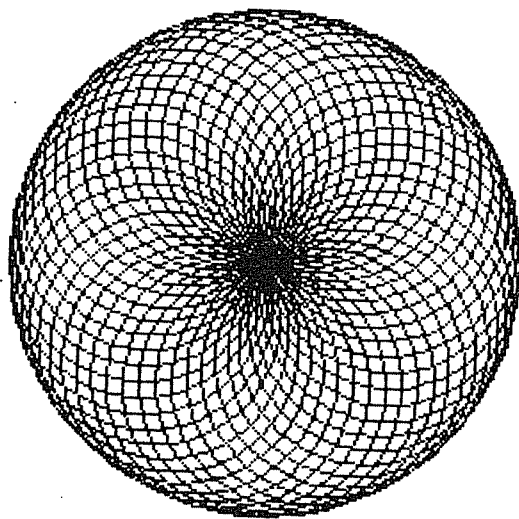


Fig. 3.27

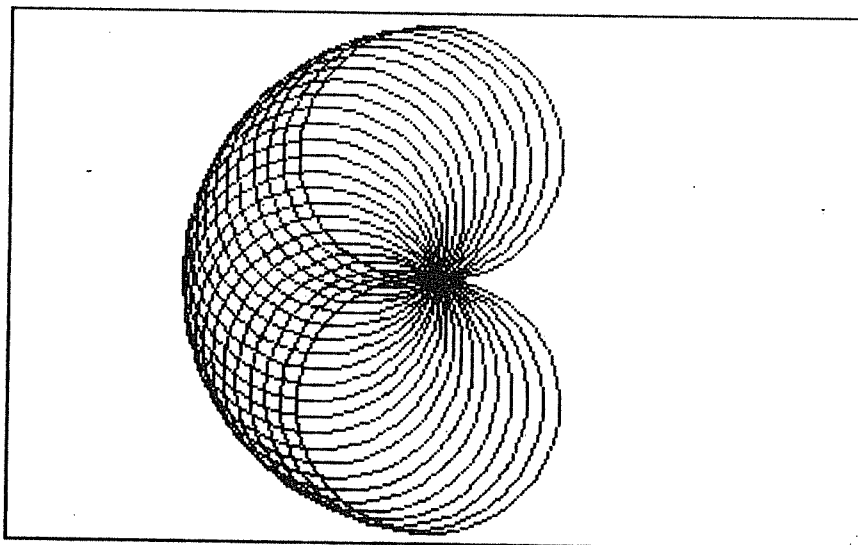


Fig. 3.28

```

0  *** 3-28 : Half a shell **
10 SCREEN 1:COLOR 0,1:CLS
20 T=95
30 FOR A=0 TO 78.65 STEP .05
35   R=T*SIN((A+F)*.8)
40   X=160+R*COS(A)

```

```

50  Y=100+R*SIN(A)
60  IF A=0
      THEN
        PSET(X,Y)
      ELSE
        LINE-(X,Y)
65  F=F+.01
70  NEXT

```

PROGRAMME 3.28

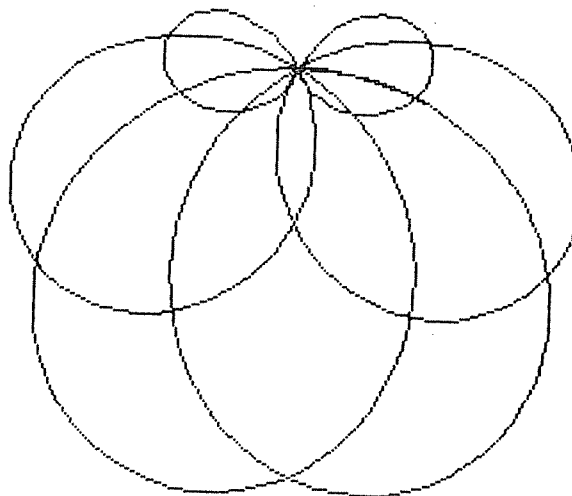


Fig. 3.29

```

0  "" 3-29 : Rug beater ""
10 SCREEN 1:COLOR 0,1:CLS
20 T=85
30 FOR A=0 TO 16 STEP .05
35  R=T*(SIN(A+F)+SIN(A) )
40  X=160+R*COS(A)
50  Y=30+R*SIN(A)
60  IF A=0
      THEN
        PSET(X,Y)
      ELSE
        LINE-(X,Y)

```

```

65 F=F+.02
70 NEXT

```

PROGRAMME 3.29

```

0  '° 3-30 : Escargot °°
10 SCREEN 1:COLOR 0,1:CLS
20 T=85
30 FOR A=0 TO 29.8 STEP .05:' (or 0-63)
35  R=T*(SIN(A+F)+SIN(A))
40  X=160+R*COS(A)
50  Y=30+R*SIN(A)
60  IF A=0
      THEN
        PSET(X,Y)
      ELSE
        LINE-(X,Y)
65  F=F+5.000001E-03
70 NEXT

```

PROGRAMME 3.30

Si on change la ligne 30 du programme 3.30 en :

```

30 FOR A=0 TO 63 STEP .05

```

on obtient l'autre moitié de la forme comme le montre la figure 3.31.

Si on dessine un cercle en coordonnées polaires et que le rayon croisse au fur et à mesure que l'angle croît, c'est une hélice qui sera dessinée. Dans le programme 3.9 nous utilisions des demi-cercles pour dessiner une hélice. Ici, comme la forme est ajustée à chaque point la figure sera plus proche de l'hélice idéale. Le programme 3.31 dessine l'hélice de la figure 3.32.

```

0  '° 3-31 : Progressive spiral °°
10 SCREEN 1:CLS
20 FOR I=0 TO 45 STEP .1
30  R=I^2
40  X=R*COS(I)
50  Y=R*SIN(I)
60  IF I=0
      THEN
        PSET(160+X,100+Y)
      ELSE

```

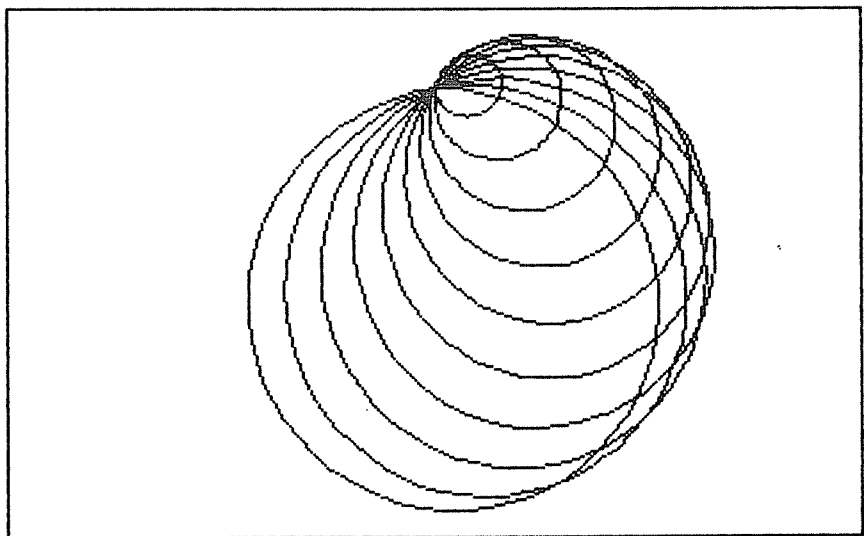


Fig. 3.30

```

LINE-(160+X,100+Y)
70 NEXT

```

PROGRAMME 3.31

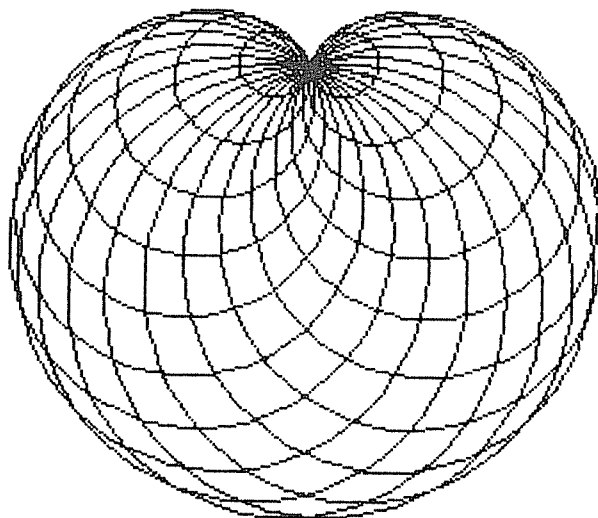


Fig. 3.31

Aux paragraphes 8.3.2 et 8.4 nous présenterons une technique pour dessiner toute une nouvelle variété de courbes très proches de celles

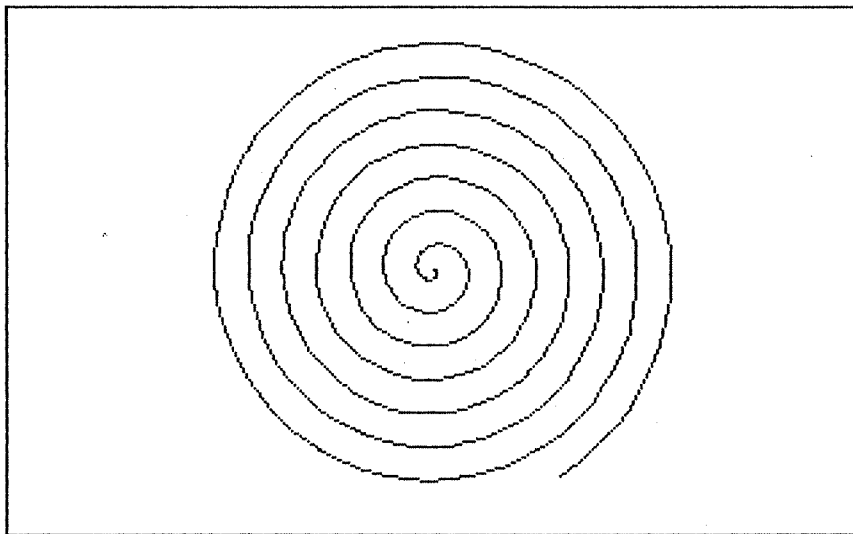


Fig. 3.32

de ce paragraphe, y compris des hélices qui ne sont pas orientées verticalement ou horizontalement.

CONDITIONS LIMITES

RAYON. Le rayon utilisé dans l'instruction `CIRCLE` peut être n'importe quel nombre entre 0 et 32767. Si on utilise un nombre négatif, ce sera ce nombre, auquel on ajoute 32768, qui sera pris comme rayon. Par exemple, dans l'instruction `CIRCLE(159,100),-1`, le rayon utilisé en réalité sera $32768-1$, c'est-à-dire 32767. L'intervalle des valeurs négatives autorisées est donc de -1 à -32767 .

CENTRE. Les coordonnées du centre du cercle peuvent être n'importe quels nombres compris entre -32768 et 32767 . Bien qu'un cercle centré en un point situé en dehors de l'écran ne produise qu'un petit arc sur l'écran (si toutefois une partie du cercle tombe dans l'écran), quand on a un très grand rayon, c'est le cercle en entier qui est étudié et l'ensemble du processus peut prendre beaucoup de temps. Par exemple il faudra 15 secondes pour réaliser un cercle d'un rayon de 10000 alors qu'avec un rayon de 32767 cela prendra 50 secondes. Dans les extrêmes, les cercles peuvent parfois se comporter de façons inattendues : la suite d'instructions

```
10 SCREEN 1:COLOR 0,0
20 CIRCLE (20100,20100),32767,1
```

30 CIRCLE (20100,20100),32765,2
40 CIRCLE (20100,20100),32764,3

devrait dessiner les arcs avec les couleurs (de bas en haut) 1,2,3. Toutefois les couleurs sont affichées dans l'ordre 2,1,3, illustrant le comportement imprévisible dans les cas de rayons limites.

ASPECT. Peut être n'importe quel nombre entier positif. Toutefois n'importe quelle valeur supérieure à 512 produira une ligne verticale et non une ellipse, et n'importe quelle valeur inférieure à $1,953125E-03$ produira une ligne horizontale. Les valeurs négatives entre 0 et -128 sont interprétées comme si elles étaient entre 0 et 1, seulement elles ne sont pas distribuées régulièrement et produisent des ellipses aléatoires. Les valeurs inférieures à -128 entraînent une erreur de dépassement de capacité.

DEBUT ET FIN doivent être supérieurs ou égaux à -6,283186 et inférieurs ou égaux à 6,283186.

REVISION

CIRCLE(100,50),10	Dessine un cercle centré en (100,50) avec un rayon de 10. Comme aucune couleur n'est précisée, c'est la couleur 3 qui est utilisée par défaut.
CIRCLE STEP(10,-6),20	Dessine un cercle centré 10 pixels sur la droite et 6 pixels au dessus du LRP. Le rayon est 20 et la couleur (par défaut) 3.
CIRCLE(120,40),15,2	Dessine un cercle centré en (120,40) avec un rayon de 15 et la couleur 2.
CIRCLE(50,20),15,1,0,3	Dessine un arc de cercle centré en (50,20), rayon 15, et couleur 1. L'arc va de l'angle 0 à 3 radians.
CIRCLE(30,60),20,2,,,2	Dessine une ellipse dont le centre est en (30,60) et la couleur 2. Le rayon sur l'axe des Y est 20 et 10 sur l'axe des X.
CIRCLE(160,100),50,3,,,5	Dessine une ellipse centrée en (160,100) avec la couleur 3. Le rayon sur l'axe des X est 50 et 25 sur l'axe des Y.

EXERCICES

1. Ecrire un programme qui dessine une spirale ellipsoïdale.
2. Ecrire un programme qui dessine un monde comme celui de la figure 3.14, mais au lieu d'utiliser des ellipses horizontales et verticales concentriques, dessiner l'équivalent de parallèles et de méridiens.
3. Ecrire un programme qui dessine un rectangle incliné d'un angle C par rapport à l'horizontale.
4. Ecrire un programme qui dessine la figure 3.33.
5. Dessiner le chapeau de Charlie Chaplin.
6. Dessiner la planète Saturne avec plusieurs anneaux. Assurez-vous que les anneaux ne soient visibles que devant la planète.

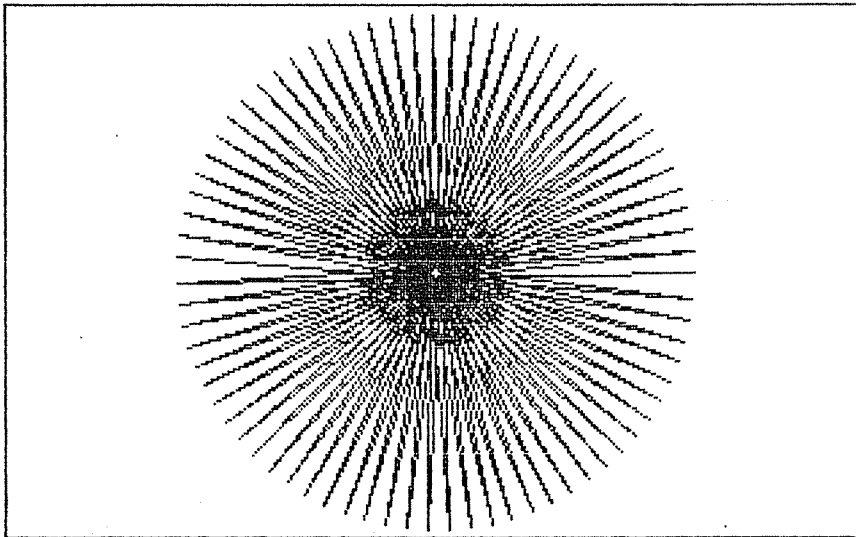


Fig. 3.33

NOTES

1. Pi en simple précision est 3,141593. Comme toutes les fonctions trigonométriques sont calculées en simple précision, il n'est pas nécessaire d'utiliser PI en double précision (3,141592653589793). $2 * \text{PI}$ est donc 6,283186.

2. Pour calculer le pourcentage d'un groupe de nombres, ajoutez les nombres pour avoir le total. On pourra avoir chaque pourcentage avec la formule
$$\text{PERCENT} = \text{NUMBER} * 100 / \text{TOTAL}$$
3. Les pixels ne sont pas parfaitement carrés sur l'IBM PC. Leur hauteur est plus grande que la largeur.

100

100

4

La couleur

Les figures en forme de fil de fer qu'on peut créer avec des lignes droites et des courbes sont appropriées pour représenter certains objets de la vie réelle. Une reproduction plus fidèle est réalisée si ces contours sont remplis avec de la couleur. Colorier un carré ou un rectangle avec une couleur particulière ne présente pas de difficulté parce que les limites sont droites et leur emplacement est connu. Malheureusement quand la ligne à colorier a une forme complexe, le processus peut devenir très difficile.

4.1 L'INSTRUCTION PAINT

BASIC fournir un moyen simple pour remplir un contour avec de la couleur : l'instruction PAINT. Sa syntaxe est PAINT (*colonne,ligne*), *couleur1,couleur2*.

Colonne et *ligne* peuvent être sous forme absolue ou relative et indiquent le point où le coloriage doit commencer. *Couleur1* indique le code couleur avec laquelle la forme doit être coloriée, et *couleur2* est le code couleur du pourtour, c'est-à-dire la couleur qui délimitera le coloriage. *Couleur1* peut être égal à *couleur2*.

Le programme 4.1 par exemple remplit un cercle jaune avec du rouge.

```
0 ** 4-1 : Fill circle **  
10 SCREEN 1:COLOR 0,0
```

```
20 CIRCLE(160,100),80,3
30 PAINT(160,100),2,3
```

PROGRAMME 4.1

Le programme 4.2 dessine une tête et colore le visage en rouge et les cheveux en vert. La figure 4.1 montre la version noir et blanc de ce visage.

```
0 ' ** 4-2 : Face **
10 SCREEN 1:COLOR 0,0:CLS
20 ' face
30 CIRCLE(160,100),99,3,,,1.1
40 ' right eye
50 CIRCLE(122,85),25,3,,,4
60 CIRCLE(122,85),12
70 CIRCLE(122,85),3,3,,,1.1
80 PAINT(125,90),3,3
90 ' left eye
100 CIRCLE(195,85),25,3,,,4
110 CIRCLE(195,85),12
120 CIRCLE(195,85),3,3,,,1.1
130 PAINT(195,90),3,3
140 ' ears
150 CIRCLE(70,90),20,3,1,4.8,2.5
160 CIRCLE(250,90),20,3,4.6,2,2.5
170 ' erase connection of ears
180 CIRCLE(160,100),99,0,2.85,3.24,1.1
190 CIRCLE(160,100),99,0,6.17,.3,1.1
200 ' nose
210 CIRCLE(160,115),21,3,,,1.3
220 ' mouth
230 CIRCLE(160,105),75,3,3.5,5.5,1.2
240 CIRCLE(175,86),90,3,3.8,5.05
250 ' hair
260 CIRCLE(100,-30),120,3,4.48,5.76
270 CIRCLE(324,-4),120,3,3.35,4
280 'color face with red
290 PAINT(160,90),2,3
300 ' color hair
310 PAINT(160,30),1,3
320 PAINT(220,40),1,3
```

PROGRAMME 4.2

Quand le point situé en (*colonne,ligne*) a déjà la couleur *color2*, le coloriage est supposé accompli et l'instruction est ignorée, comme c'est le cas dans le programme 4.3.

```
0 '°° 4-3 : PAINT ignored °°
10 SCREEN 1:CLS
20 LINE(50,50)-(150,150),3,B
30 PAINT(150,150),3,3
```

PROGRAMME 4.3

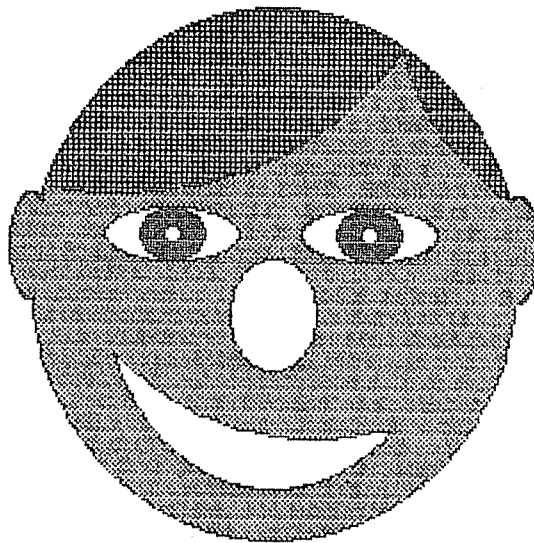


Fig. 4.1

Les couleurs différentes de *couleur2* sont complètement ignorées et remplacées par *couleur1*, comme le montre le programme 4.4.

```
0  " 4-4 : Colors replaced in PAINT "
10 SCREEN 1:CLS
20 LINE(50,50)-(150,150),2
30 LINE(50,150)-(150,50),1
40 CIRCLE(100,100),40,2
50 LINE(50,50)-(150,150),3,B
60 W$=INPUT$(1)
70 PAINT(100,100),3,3
```

PROGRAMME 4.4

Si le contour a au moins un point de discontinuité (un point d'une couleur différente) le remplissage débordera dans la zone extérieure, comme le montre le programme 4.5.

```
0  " 4-5 : PAINT escapes through the border "
10 SCREEN 1:COLOR 0,0:CLS
20 LINE(100,50)-(300,50)
30 LINE-(300,150)
40 LINE-(100,150)
50 LINE(100,148)-(100,50)
60 PAINT(160,100),2,3
```

PROGRAMME 4.5

4.2 DEPASSEMENT DE MEMOIRE PROVOQUE PAR PAINT

Dans le processus de coloriage, chaque fois qu'un embranchement est atteint (un point où le coloriage peut s'étendre dans deux directions différentes, BASIC doit mémoriser un des chemins tout en poursuivant dans l'autre. Quand il a fini avec l'un, le processus peut continuer avec ceux qui étaient différés. Pour mémoriser ceci, BASIC utilise ses blocs mémoires. Quand la figure à colorier est très complexe, le programme dépasse la capacité disponible et une erreur « *out of memory* » (dépassement de la mémoire) l'interrompt. Le programme 4.6 illustre cette situation.

```
0 *** 4-6 : Amnesia **  
10 SCREEN 1:CLS  
20 FOR I=3 TO 300 STEP 5  
30   LINE(I,100)-(I,110),3  
40 NEXT  
50 PAINT(0,100),2,3
```

PROGRAMME 4.6

Si un programme doit manipuler PAINT avec des formes complexes, il est recommandé d'augmenter la mémoire de stockage (ce qui diminuera l'espace disponible pour les variables et le programme) en utilisant l'instruction CLEAR au début du programme. Si on ajoute la ligne 5 CLEAR,, 700 au programme 4.6, l'erreur ne sera pas produite.

4.3 CERCLES ET ELLIPSES PLEINS

L'instruction CIRCLE ne peut produire que des lignes courbes, pas des cercles pleins. Toutefois avec l'instruction PAINT, il est possible de dessiner un cercle ou une ellipse et d'en colorier l'intérieur. La ligne CIRCLE(80,80),70,2:PAINT(80,80),2,2, par exemple dessine un cercle plein centré en (80,80). Une autre manière d'arriver au même résultat sera CIRCLE(80,80),70,2:PAINT STEP(0,0),2,2, parce que CIRCLE laisse le LRP en son centre, et que c'est le point le plus sûr pour commencer à colorier¹. Cette syntaxe présente de plus l'avantage d'éliminer les erreurs dans les coordonnées du centre.

Bien que ces deux suites d'instructions puissent être considérées à première vue comme l'équivalent circulaire de la forme pleine obtenue à partir de l'instruction LINE, en fait elles ne garantissent pas vraiment que le cercle soit rempli complètement. Si le centre du cercle a été précédemment tracé avec la couleur du cercle, comme dans la ligne PSET (90,90),2:CIRCLE(90,90),80,2:PAINT STEP(0,0),2,2,PAINT n'aura aucun effet. Si des parties de la zone à l'intérieur du cercle ont des formes

de la même couleur, elles ne seront pas effacées par PAINT, comme dans l'instruction `LINE(0,100)-(155,100),2:CIRCLE(160,100),80,2:PAINT(160,100),3,2`. Si ces formes encombrant le chemin de PAINT, toute une zone à l'intérieur du cercle ne sera pas coloriée du tout, comme il est montré dans le programme 4.7.

```
0  "" 4-7 : Path of PAINT obstructed ""
5 SCREEN 1:CLS
10 LINE(0,99)-(160,99),1:LINE(161,99)-(161,199),1
20 CIRCLE(160,100),80,1:PAINT STEP(0,0),1,1
```

PROGRAMME 4.7

Nous allons vous montrer trois méthodes pour s'assurer que le cercle ou l'ellipse soit rempli complètement de la couleur désirée.

La première, que nous appellerons méthode de coloriage « bestiale » est basée sur le fait que des coloriages successifs effectués avec différentes couleurs effacent les formes indésirables et finalement le coloriage fait son chemin jusqu'à la périphérie. Cette méthode est efficace quelle que soit la complexité des formes contenues dans le cercle car, à chaque fois qu'un coloriage est réalisé avec la couleur X comme limite, toutes les formes ayant des couleurs différentes de X sont effacées.

Dans le programme 4.8, le but est de colorier le cercle dessiné par l'instruction 80. Les instructions 20 à 50 font de la zone intérieure du cercle l'une des plus difficiles à colorier en la remplissant avec des cercles concentriques, chacun d'une couleur différente. Toute tentative pour colorier le cercle en partant de n'importe quel point intérieur n'ira pas très loin. La boucle des instructions 60 à 90 répète un coloriage qui prend son origine au centre, chaque fois avec une couleur différente. Le cercle extérieur doit être redessiné dans la couleur du contour, ou sinon, quand les obstacles seront finalement enlevés, le coloriage pourra déborder sur la zone extérieure au cercle.

```
0  "" 4-8 : Hard to flood circle ""
10 SCREEN 1:CLS
20 FOR RADIUS=1 TO 108 STEP 2
30   CIRCLE(160,100),RADIUS,COL
40   COL=COL+1:
      IF COL=4
      THEN
        COL=1
50 NEXT
60 FOR I=1 TO 31
70   C=I MOD 4
80   CIRCLE(160,100),108,C
```



```
85 PAINT STEP(0,0),C,C
90 NEXT
```

PROGRAMME 4.8

La seconde méthode, que nous appellerons la méthode par « ligne perforante » est plus rapide et plus élégante que la première mais pas efficace à cent pour cent. Il faut aussi avoir une connaissance parfaite du contour. L'idée est de tracer une ligne à travers le cercle (un diamètre) avec une couleur différente de celle avec laquelle le cercle doit être rempli. N'importe quelle forme traversée par cette ligne est perforée, et un simple PAINT est suffisant pour la pénétrer. Le programme 4.9 colore le cercle problème du programme 4.8 seulement en une fraction du temps nécessaire avec la méthode « bestiale ».

```
0  " 4-9 : Puncturing line "
10 SCREEN 1:CLS
20 RADIUS=95:COL=2
25 CNTRX=160:CNTRY=100
30 FOR R=1 TO RADIUS STEP 2
40  CIRCLE(CNTRX,CNTRY),R,COL,,,1
50  COL=COL+1:
    IF COL=4
    THEN
      COL=1
60 NEXT
70 X=RADIUS*COS(0)
80 LINE(CNTRX-X,CNTRY)-(CNTRX+X,CNTRY),COL+1 MOD 4
90 CIRCLE(CNTRX,CNTRY),RADIUS,COL,,,1
100 PAINT(CNTRX,CNTRY),COL,COL
```

PROGRAMME 4.9

Il existe des cas pour lesquels cette méthode ne marchera pas correctement, parce que certaines formes fermées ne sont pas atteintes par la ligne perforante. Si la portion du programme 4.10 est incluse dans le programme 4.9, une partie du cercle ne sera pas colorée, comme le montre la figure 4.2.

```
62 FOR R=2 TO 52 STEP 2
64  CIRCLE(160,50),R,C MOD 4:C=C+1
66 NEXT
```

PROGRAMME 4.10

On peut utiliser une combinaison des méthodes bestiale et par ligne perforante pour plus de vitesse et de précision. Par exemple quatre lignes

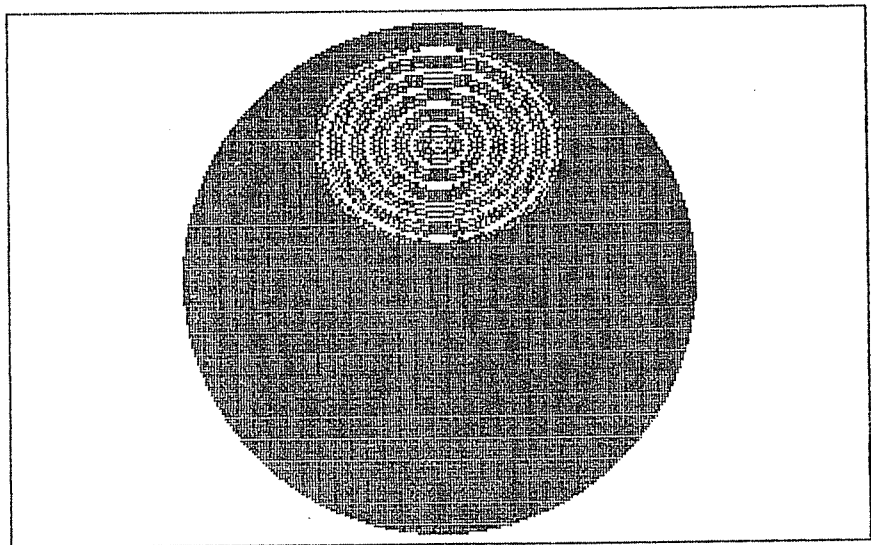


Fig. 4.2

perforantes permettront la pénétration de la plupart des formes, et le peu de zones qui ne sont pas touchées par les lignes peuvent être éliminées par quelques itérations selon la méthode bestiale. Le calcul de ces lignes peut être difficile quand *aspect* est différent de 1 et il faut trouver un compromis entre le nombre de lignes et le nombre d'itérations de la méthode bestiale.

Dans la troisième méthode nous utiliserons l'équation du cercle (voir paragraphes 2.5.4) pour trouver des points symétriques qui peuvent être reliés par des droites, produisant donc un véritable cercle plein. Vous vous rappelez que le programme 2.11 utilisait chaque point pour tracer les parties supérieure et inférieure du cercle. Si on connecte chaque paire de points symétriques par une droite, tout ce qui se trouve entre eux sera effacé. Le programme 4.11 trace un cercle plein avec *aspect*1. Le centre et le rayon doivent être introduits aux lignes 20 et 30.

```
0  "" 4-11 : Flooding circle with equation ""
10 SCREEN 1:CLS
20 INPUT"Center (X,Y) ";CNTRX,CNTRY
30 INPUT"Radius ";R
40 CLS
50 FOR X=-R TO R
60   Y=SQR(R*R-X*X)
70   LINE(X+CNTRX,CNTRY+Y)-(X+CNTRX,CNTRY-Y)
80 NEXT
```

PROGRAMME 4.11

Cette méthode marche bien avec les cercles « normaux », mais comme elle calcule et trace chaque ligne sans se soucier si elle est à l'intérieur de l'écran ou non, de très grands cercles peuvent être trop longs à achever. Par exemple un cercle centré en (160,100) et avec un rayon de 1000 prend 1 minute 36 secondes pour être tracé. Le même cercle avec un rayon de 32767 prendra plus d'1 heure et demie pour être terminé ! Nous savons que $-R$ se trouve à gauche du centre et R à sa droite (nous supposons que les rayons sont positifs), ainsi nous pouvons vérifier si ces deux points sont à l'extérieur de l'écran et les traiter en conséquence. Dans le programme 4.12, la ligne 42 vérifie si $-R$ est inférieur à 0 après avoir été ajusté au centre précisé. Si c'est le cas, START est mis à 0, sinon il est réglé à $-R$. De la même manière la ligne 44 vérifie si R produira des lignes à l'extérieur de l'écran et ajuste la variable ENDING en conséquence. Avec ces deux valeurs, la boucle FOR ne peut utiliser que des valeurs qui produisent des lignes à l'intérieur de l'écran.

```

0  ° 4-12 : Flooded circle: limits adjusted °
10 SCREEN 1:CLS
20 INPUT"Center (X,Y) ";CNTRX,CNTRY
30 INPUT"Radius ";R
40 CLS
42 IF -R+CNTRX < 0
    THEN
        START=-CNTRX
    ELSE
        START=-R
44 IF R+CNTRX > 319
    THEN
        ENDING=319-CNTRX
    ELSE
        ENDING=R
50 FOR X=START TO ENDING
60   Y=SQR(R²-X²)
70   LINE(X+CNTRX,CNTRY+Y)-(X+CNTRX,CNTRY-Y)
80 NEXT

```

PROGRAMME 4.12

Il y a une dernière touche à rajouter pour que ce programme soit complet : quand des parties du cercle sont en dessus ou en dessous de l'écran, une ligne est tracée en bordure de l'écran. La figure 4.3 montre le cercle tracé par le programme 4.12 avec un centre à (160,300) et un rayon de 150.

Remarquez les deux lignes sur les côtés du cercle. Pour corriger ce problème, nous pouvons vérifier si le bas de la ligne est au-dessus de l'écran



Fig. 4.3

ou si le haut de la ligne est en dessous. Si on ajoute la ligne suivante au programme 4.2, elle s'occupera des « lignes étrangères ».

```
62  IF CNTRY+Y<0 OR CNTRY-Y>199
    THEN
        80
```

4.4 POINTS A L'EXTERIEUR DE L'ECRAN

On ne peut pas utiliser PAINT avec des points situés au dehors des intervalles autorisés. Si au moins une des coordonnées du point où le coloriage doit commencer tombe à l'extérieur de l'écran, c'est le point valide le plus proche qui sera utilisé à la place. Le programme 4.13 calcule les valeurs autorisées les plus proches à la fois pour les coordonnées des lignes et des colonnes, et colore n'importe quel cercle autorisé.

```
0  ** 4-13 : Adjust parameters for PAINT **
10 SCREEN 1:COLOR 0,0:CLS
20 INPUT"Center (X,Y) ";X,Y:CLS
30 CIRCLE(X,Y),80,2,2
40 GOSUB 1000:GOSUB 1100
50 PAINT(X,Y),2,2
60 END
1000 ' Return in X the nearest valid value
1010 IF X<0
    THEN
        X=0:RETURN
    ELSE
        IF X>319
            THEN
                X=319:RETURN
            ELSE
                RETURN
1100 ' Return in Y the nearest valid value
1110 IF Y<0
    THEN
```

```

Y=0:RETURN
ELSE
  IF Y>199
  THEN
    Y=199:RETURN
  ELSE
    RETURN

```

PROGRAMME 4.13

4.5 POLYGONES PLEINS

Les polygones peuvent être colorés avec les méthodes pour les cercles et les ellipses expliquées dans le paragraphe précédent, parce qu'un polygone peut être considéré comme un cercle dessiné en basse résolution. Comme pour les cercles, si le centre du polygone est en dehors de l'écran, les paramètres de PAINT devront être changés et les valeurs autorisées les plus proches seront utilisées comme dans la méthode expliquée au paragraphe précédent. La figure 4.4 montre un octogone plein.

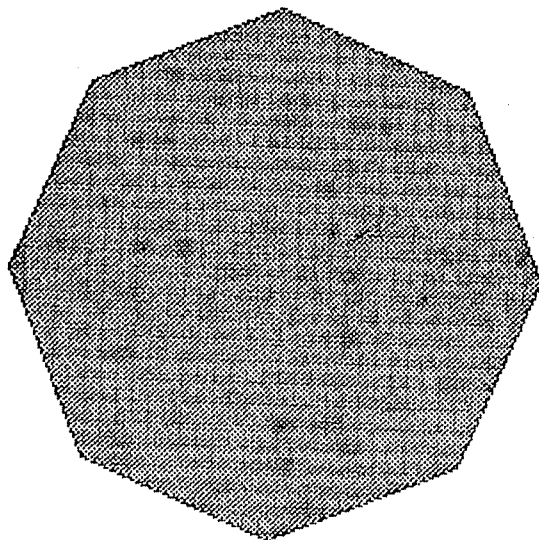


Fig. 4.4

4.6 CAMEMBERTS PLEINS

Une limitation sérieuse pour les camemberts, expliquée au paragraphe 3.2.4, est que seul le contour des morceaux est dessiné, et comme les

séparations se chevauchent, elles s'effacent mutuellement, le résultat final n'étant pas très clair.

Des camemberts pleins, avec des parts colorées différemment, donnent une idée beaucoup plus précise de l'importance de chaque partie par rapport aux autres. Malheureusement le choix du point de départ du coloriage n'est pas aussi facile que pour les rectangles, les cercles ou les polygones. On ne peut pas prendre le centre du camembert et trouver un point près du contour est compliqué. Ceci est l'un des cas pour lequel le concept de coordonnées polaires appris au paragraphe 3.4 prouvera son utilité.

La ligne 170 du programme 4.14 trace le camembert comme le faisait le programme 3.11. *A* est le *début* et *A + V* est la *fin* de chaque arc. La moyenne des deux, $(A + (A + V)) / 2$ est le point équidistant entre *début* et *fin*. Aux lignes 190 et 200, le point où le coloriage doit commencer est calculé en prenant l'angle moyen et les équations 3.1 et 3.2. Toutefois le rayon utilisé ne peut être le même que celui du cercle, ou sinon le point calculé se trouverait exactement sur la périphérie, et PAINT n'aurait aucun effet. Un bon choix pour le rayon sera la moitié du rayon du cercle. Comme le programme 4.14 ne précise aucun *aspect*, 5/6 sera fixé par défaut et c'est ce facteur qui doit être utilisé pour calculer les coordonnées Y de l'origine de PAINT. La figure 4.15 montre un diagramme en forme de gâteau produit par le programme 4.14.

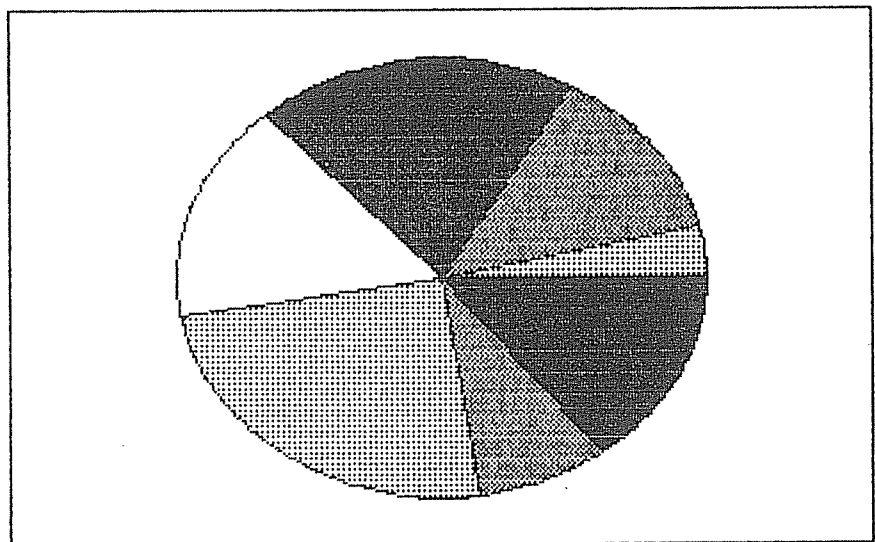


Fig. 4.5

```
0  ** 4-14 : Solid pie wedges **
10 SCREEN 1:CLS:DIM M(20),T$(20)
20 DEF FN ANG(X)=2*PI*X/100
```

```

30 PI=3.141593:A=0:F=1.745329E-02
40 CNTRX=160:CNTRY=100
50 RADIUS=98:ASPECT=5/6
60 INPUT"Number of items ";N
70 FOR I=1 TO N:PRINT"amount ";
80   INPUT M(I)
90 NEXT
100 FOR I=1 TO N:
    T=T+M(I):
NEXT
110 CLS
120 FOR I=1 TO N:
    M(I)=M(I)*100/T:
NEXT
130 FOR I=1 TO N
140   V=FN ANG(M(I))
150   COL=I MOD 4
160   IF A+V>6.283186
    THEN
        A=6.283186-V
170   CIRCLE(CNTRX,CNTRY),RADIUS,COL,-A,-(A+V)
180   ANG=(A+A+V)/2
190   X=CNTRX+RADIUS*.5*COS(ANG)
200   Y=CNTRY+RADIUS*.5*SIN(ANG)*ASPECT
210   PAINT(X,200-Y),COL,COL
220   CIRCLE(CNTRX,CNTRY),RADIUS,3,-A,-(A+V)
230   A=A+V
240 NEXT

```

PROGRAMME 4.14

4.7 LES COULEURS NON STANDARDS

Quatre couleurs seulement peuvent être affichées en moyenne résolution, mais beaucoup d'autres peuvent être créées par combinaison. Si au lieu d'utiliser des pixels on pense à un bloc plus gros de quatre pixels comme unité de base, beaucoup de combinaisons de couleurs seront possibles. Si on suppose que l'écran a été réglé avec l'instruction COLOR 1,0, la figure 4.6 montre les cinq différentes couleurs qui peuvent être générées en combinant les couleurs 0 et 2.

Quand on règle le fond et la palette avec COLOR 1,0 (ou COLOR 9,0) on obtient la plus grande variété parce que toutes les couleurs de base sont prédéfinies : bleu (fond), vert, rouge et jaune.

Le programme 4.15 remplit l'écran avec des blocs de 12*14 pixels, produisant toutes les combinaisons dans un bloc de base de 2*2 pixels. Comme il y a quatre pixels dans le bloc de base et quatre couleurs possibles, le total des différentes combinaisons possibles est 255. Toutefois les couleurs obtenues ne sont pas toutes distinctes et du fait des symétries beaucoup sont répétées.

```

0  " 4-15 : Variety of colors "
10 SCREEN 1:COLOR 1,0:CLS

```

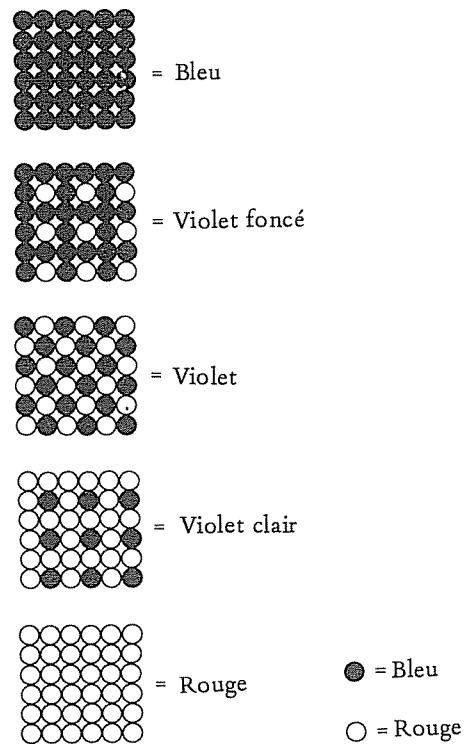


Fig. 4.6

```

20 FOR C1=0 TO 3
30   FOR C2=0 TO 3
40     FOR C3=0 TO 3
50       FOR C4=0 TO 3
60         GOSUB 1000
65         X=X+14:
           IF X>300
             THEN
               X=0:Y=Y+16
70       NEXT:
72     NEXT:
74   NEXT
76 NEXT
80 END
1000 ' °° Draw one 12 ° 14 clock °°
1010 FOR Q=X TO X+12 STEP 2
1020   FOR W=Y TO Y+14 STEP 2
1030     GOSUB 2000
1040   NEXT

```



```

1050 NEXT
1060 RETURN
2000 ' ** Draw the basic 2 * 2 block **
2010 PSET(Q,W),C1:PSET(Q,W+1),C2:
2020 PSET(Q+1,W),C3:PSET(Q+1,W+1),C4
2030 RETURN

```

PROGRAMME 4.15

Le sous-programme de la ligne 2000 produit le véritable bloc de base de quatre pixels. Pour utiliser ces nouvelles couleurs et éviter les répétitions, il faut choisir certaines combinaisons et mémoriser leurs codes-couleurs dans un tableau à deux dimensions, comme le montre le programme 4.16. Les quatre différentes valeurs dont le sous-programme de la ligne 2000 a besoin pour chaque couleur seront A(I,1), A(I,2), A(I,3) et A(I,4). L'index I indiquera le numéro de couleur dans la nouvelle palette. Un bloc de couleur pleine est obtenu quand tous les A(I,J) sont égaux pour un I précis.

```

0 ' ** 4-16 : Palette in array **
10 DEFINT A-Z: DIM A(20,4)
20 FOR I=1 TO 20
30   FOR J=1 TO 4
40     READ A(I,J)
50   NEXT
60 NEXT
70 SCREEN 1: COLOR 1,0: CLS
90 X=0: Y=0: I=1
100 WHILE X<314
110   GOSUB 1000
120   X=X+14: I=I+1
125   IF I=21
       THEN
         END
130 WEND
140 Y=Y+16: X=0: GOTO 100
1000 ' Draw a 12 * 14 block
1010 FOR Q=X TO X+12 STEP 2
1020   FOR W=Y TO Y+14 STEP 2
1030     GOSUB 2000
1040   NEXT
1050 NEXT
1060 RETURN
2000 ' Draw basic 2*2 block
2010 PSET(Q,W),A(I,1):PSET(Q+1,W),A(I,2)
2020 PSET(Q,W+1),A(I,3):PSET(Q+1,W+1),A(I,4)
2030 RETURN
10000 DATA 1,0,0,0,0,1,1,0,1,1,0,1,1,1,1,1
10010 DATA 2,0,0,0,0,2,2,0,2,2,0,2,2,2,2,2
10020 DATA 3,0,0,0,0,3,3,0,3,3,0,3,3,3,3,3
10030 DATA 2,1,1,1,1,2,2,1,2,2,1,2,3,2,2,2
10040 DATA 2,3,3,2,3,3,2,3,0,0,0,0,0,1,2,3

```

PROGRAMME 4.16

4.8 BOITES PLEINES AVEC LA PALETTE ETENDUE

Dans la palette présentée au paragraphe précédent, un bloc de deux pixels de côté était considéré comme le plus petit élément d'image. Quand toutes les formes sont dessinées avec cette taille de bloc, colorier des zones avec la palette étendue peut être réalisé en dessinant simplement le bloc entre les limites. Le programme 4.17, avec des parties du programme 4.16, dessine un rectangle avec ces blocs plus grands et remplit la zone avec la couleur choisie à la ligne 90. On doit rajouter au programme 4.17 le sous-programme de la ligne 2000 et les lignes 5 à 80 (la partie qui lit les couleurs) du programme 4.16.

```
90 INPUT "Color code (frame,fill) "; C1,C2:
   IF C1>19 OR C2>19
      THEN
         90
95 CLS
100 ' Draw a rectangle
105 I=C1
110 FOR Q=20 TO 90 STEP 2
120   W=10:GOSUB 2000
130   W=40:GOSUB 2000
140 NEXT
150 FOR W=12 TO 38 STEP 2
160   Q=20:GOSUB 2000
170   Q=90:GOSUB 2000
180 NEXT
190 ' Fill the rectangle
200 I=C2
210 FOR Q=22 TO 88 STEP 2
220   FOR W=12 TO 38 STEP 2
230     GOSUB 2000
240   NEXT
250 NEXT
260 END
```

PROGRAMME 4.17

4.9 PSET AVEC LA PALETTE ETENDUE

La méthode pour colorier utilisée dans le programme 4.17 ne peut pas être appliquée quand la bordure ne coïncide pas avec un bloc de deux pixels. Du fait que tous les blocs de deux pixels sont dessinés dans une colonne ayant un nombre pair, quand il n'y a pas assez de place pour le dernier bloc, il faut laisser un espace.

Une autre manière de tracer les blocs de deux pixels à certains emplacements consistera à tracer chaque point du bloc avec la couleur de la nouvelle palette, mais sans se soucier des autres points du bloc. On peut faire ceci en utilisant l'opérateur modulo ² (MOD). Tout nombre pair MOD 2 est égal à zéro (p. ex. $34 \text{ MOD } 2 = 0$; $124 \text{ MOD } 2 = 0$) et tout nombre impair positif MOD 2 est égal à 1. (p. ex : $1 \text{ MOD } 2 = 1$; $189 \text{ MOD } 2 = 1$).

La figure 4.7 montre une partie de l'écran où on peut voir comment chaque point peut être réduit à un des quatre pixels du tableau à deux dimensions qui contient les couleurs de la nouvelle palette, comme cela est fait dans le programme 4.16. Chaque cellule représente un pixel avec ses coordonnées en haut et un couple de nombres en bas. Ces nombres indiquent si la coordonnée correspondante est paire (0) ou impaire (1). On peut remarquer que les coordonnées de chaque point déterminent sa position dans le bloc 2×2 .

Nous allons changer la représentation des couleurs dans la palette étendue en $M(\text{col}, \text{guidocol}, \text{guideligne})$, où *col* est le code couleur étendu (1 quel que soit le nombre de couleurs choisies), *guidocol* est 0 si le numéro de colonne est un nombre pair et 1 si non et *guideligne* est 0 ou 1 selon la parité du numéro de ligne.

Les indices du point (4,3), par exemple, sont $4 \text{ MOD } 2$ (0); et $3 \text{ MOD } 2$ (1). Avec ce nouveau schéma, la couleur de ce point peut être trouvée dans $M(\text{COL}, 0, 1)$, c'est-à-dire la couleur pour le point dont le numéro de colonne est impair et le numéro de ligne pair, et de couleur COL. Le sous-programme de la ligne 2000 du programme 4.18 trace la couleur de la palette des blocs à deux pixels de n'importe quel point de l'écran. La ligne 100 trace un rectangle qui poserait un problème si on le coloriait en utilisant la technique du programme 4.17 et les lignes 210 à 250 le remplissent avec la couleur rentrée à la ligne 90. Remarquez que la partie active du sous-programme de la ligne 2000 a été réduite à une simple ligne.

```
0  ** 4-18 : PSET in extended palette **
10 DEFINT A-Z: DIM A(20,1,1)
20 FOR I=1 TO 20
30   FOR J=0 TO 1
35     FOR K=0 TO 1
40       READ A(I,J,K)
45     NEXT
50   NEXT
60 NEXT
70 SCREEN 1: COLOR 1,0:CLS
90 INPUT "Color code ";C1
95 CLS
100 LINE(21,11)-(91,40),2,B
200 I=C1
210 FOR Q=22 TO 90
220   FOR W=12 TO 39
230     GOSUB 2000
240   NEXT
250 NEXT
260 END
```

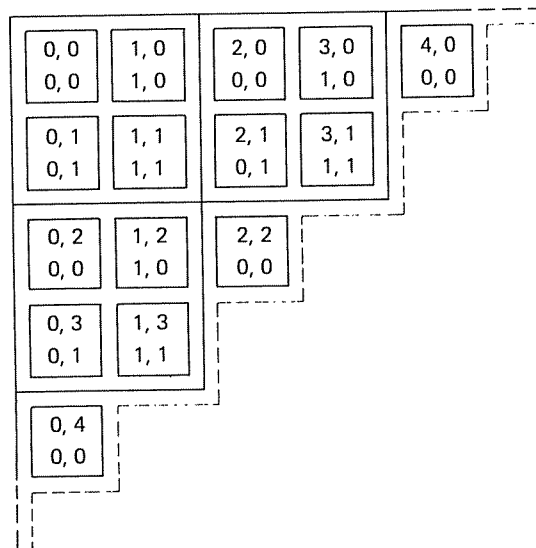


Fig. 4.7

```

2000 ' PSET with extended palette **
2010 PSET(Q,W),A(I,Q MOD 2,W MOD 2)
2030 RETURN
10000 DATA 1,0,0,0,0,0,1,1,0,1,1,0,1,1,1,1,1
10010 DATA 2,0,0,0,0,0,2,2,0,2,2,0,2,2,2,2,2
10020 DATA 3,0,0,0,0,0,3,3,0,3,3,0,3,3,3,3,3
10030 DATA 2,1,1,1,1,1,2,2,1,2,2,1,2,3,2,2,2
10040 DATA 2,3,3,2,3,3,2,3,0,0,0,0,0,0,1,2,3

```

PROGRAMME 4.18

4.10 LE TRACE DE LIGNES AVEC DES COULEURS NON STANDARDS

Au paragraphe 2.6 nous avons étudié deux méthodes pour tracer des lignes avec PSET. Si au lieu de tracer tous les points de la même couleur nous utilisons notre propre sous-programme de traçage de points (le sous-programme 2000 du programme 4.18) nous pourrions tracer des lignes avec n'importe quelle couleur de la palette étendue.

Le programme 4.19 demande une couleur à la ligne 110, obtient les coordonnées des deux points limites aux lignes 120 et 130 et les relie avec une ligne droite de la couleur choisie, si on le fusionne avec le sous-programme 2000 et la partie initialisation du programme 4.18 ; les lignes de 0 à 70.

```

110 INPUT"Color code (1-19) ";C
120 INPUT"First point (X1,Y1) ";X1,Y1
130 INPUT"Second point (X2,Y2) ";X2,Y2
140 DX=X2-X1:DY=Y2-Y1
150 D=SQR(DX*DX+DY*DY)
160 SX=DX/D:SY=DY/D
170 FOR I=0 TO D
180   X=X1+I*SX
190   Y=Y1+I*SY
192   GOSUB 2000
195 NEXT
200 END

```

PROGRAMME 4.19

4.11 LES LIGNES LARGES

La belle ligne produite par l'instruction LINE est très pratique, mais très souvent on a besoin de lignes plus larges. Grâce à notre connaissance sur le tracé des lignes point par point, nous pouvons dessiner chaque couple X,Y, non pas comme un point (PSET) mais comme un bloc plein. Le programme 4.20 trace une ligne de couleur 3 avec l'épaisseur rentrée à la ligne 35.

```

0  "" 4-20 : Wide lines ""
10 SCREEN 1:COLOR 0,0:CLS
20 INPUT"First point (X1,Y1) ";X1,Y1
30 INPUT"Second point (X2,Y2) ";X2,Y2
35 INPUT"Width ";W:W=W-1
40 DX=X2-X1:DY=Y2-Y1
50 D=SQR(DX*DX+DY*DY)
60 SX=DX/D:SY=DY/D
70 FOR I=0 TO D
80   X=X1+I*SX
90   Y=Y1+I*SY
92   LINE(X,Y)-(X+W,Y+W),3,BF
95 NEXT

```

PROGRAMME 4.20

La figure 4.8 montre plusieurs lignes dessinées par le programme 4.20, avec des épaisseurs allant de 1 à 10.

Si la ligne 92 est remplacée par n'importe laquelle des méthodes expliquées dans les paragraphes précédents pour avoir accès à la palette étendue, alors la ligne sera dessinée dans n'importe laquelle des nouvelles couleurs.

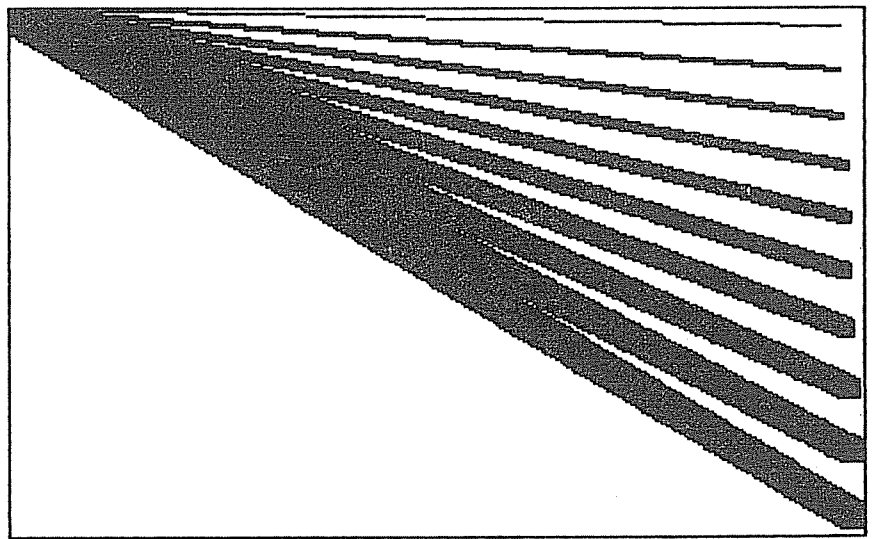


Fig. 4.8

4.12 EXTENSION DE LA PALETTE ETENDUE

En utilisant la technique de l'opérateur MOD expliquée au paragraphe 4.9, on peut définir des palettes encore plus grandes que celles des blocs 2×2 . Si par exemple au lieu d'utiliser MOD 2, le programme utilise MOD 3, il y aura neuf emplacements différents dans chaque bloc, au lieu des quatre que nous avons utilisés jusqu'à maintenant. La figure 4.9 montre une des dix manières possibles d'arranger 2 couleurs pour produire différentes concentrations de couleurs. Cet arrangement est purement arbitraire.

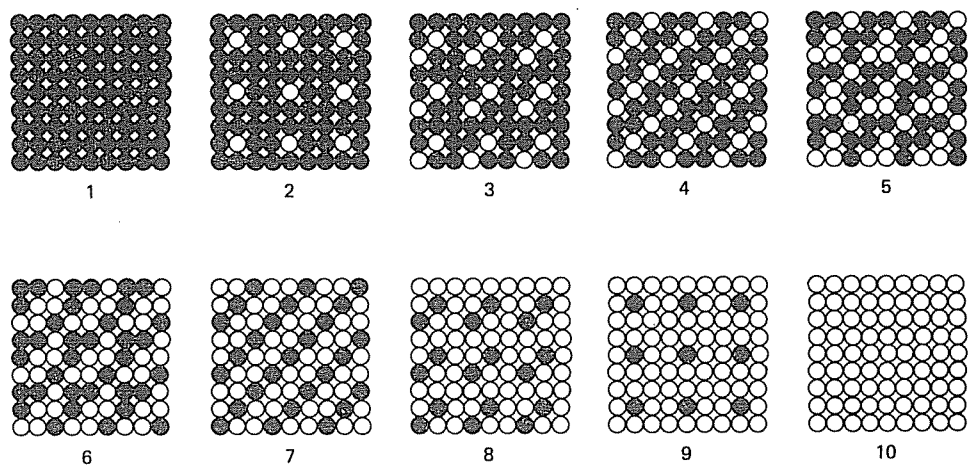


Fig. 4.9

traire et la principale chose à considérer quand on le crée est le nombre de points de chaque couleur dans un bloc. Certains arrangements seront moins efficaces que d'autres.

La figure 4.10 montre un mauvais et un bon arrangement de couleurs. Supposons que nous ayons affaire à du rouge et du bleu. Dans le mauvais les trois points rouges étaient placés sur une ligne. Quand une zone est remplie avec ce bloc de base, le résultat est un ensemble de lignes verticales qui ne se mêlent pas. Dans la bonne distribution, les points étaient placés d'une manière telle que des blocs contigus ne sont pas de la même couleur. Ceci facilite la fusion des couleurs en une nouvelle, composée.

Le programme 4.21 remplit dix rectangles avec les différentes nuances de rouge et de bleu (avec le réglage COLOR 1,0) possibles avec ce bloc à neuf pixels. La figure 4.11 montre ces blocs colorés en noir et blanc.

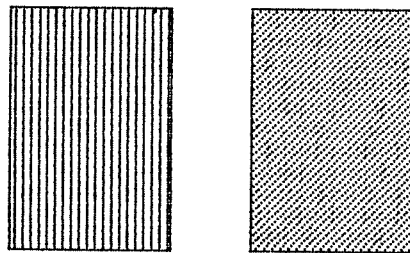


Fig. 4.10

```
0  ' ** 4-21 : 3 x 3 block **
5  CLS
10 SCREEN 1:COLOR 1,0
20 DIM M(10,2,2)
30 FOR COL=1 TO 10
40   FOR X=0 TO 2
50     FOR Y=0 TO 2
60       READ M(COL,X,Y)
70     NEXT Y
80   NEXT X
90 NEXT COL
100 FOR COL=1 TO 10
110 FOR I=0 TO 32
120   FOR J=0 TO 40
130     PSET(I+(COL-1)*32,J),M(11-COL,I MOD 3,J MOD 3)
140   NEXT J
150 NEXT I
160 NEXT COL
170 FOR I=0 TO 300 STEP 32
180   LINE(I,0)-(I+31,41),1,B
190 NEXT I
1000 DATA 0,0,0,0,0,0,0,0,0,0
1010 DATA 0,0,0,0,2,0,0,0,0,0
1020 DATA 0,0,2,0,0,0,2,0,0,0
1030 DATA 2,0,0,0,0,2,0,2,0,0
1040 DATA 2,0,0,0,2,2,0,2,0,0
1050 DATA 2,0,2,0,2,2,0,2,0,0
1060 DATA 2,0,2,0,2,2,2,2,0,0
```

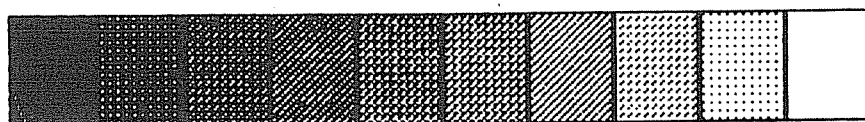


Fig. 4.11

```

1070 DATA 2,0,2,2,2,2,2,2,0
1080 DATA 2,2,2,2,0,2,2,2,2
1090 DATA 2,2,2,2,2,2,2,2,2

```

PROGRAMME 4.21

Avec les blocs 3×3 il est possible de créer un grand nombre de couleurs. Avec la combinaison de seulement deux couleurs, 44 couleurs différentes sont possibles comme le montre la figure 4.12. Quand trois ou quatre couleurs différentes sont combinées dans le bloc, le nombre de combinaisons est immense.

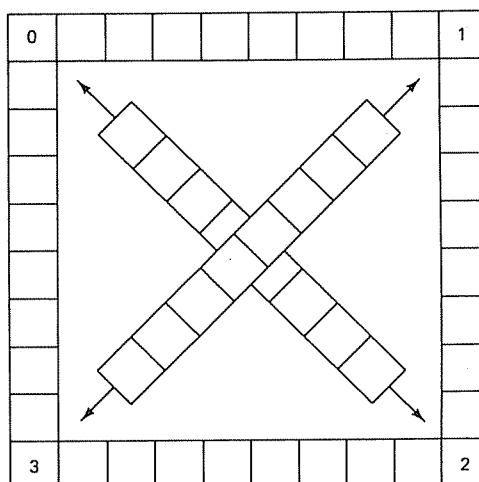


Fig. 4.12

4.13 COLORIAGE HORIZONTAL ET VERTICAL

Dans les blocs 2×2 du paragraphe précédent, quand trois pixels avaient la même couleur et qu'un était différent, le motif ressemblait à une grille. Une manière de distribuer les couleurs plus régulièrement (bien que le nombre de points par unité de surface sera la même et par conséquent la couleur aussi) est d'utiliser une grille diagonale. Le côté gauche de la figure 4.13 montre une partie de la grille horizontale, alors que le côté droit montre la grille diagonale.

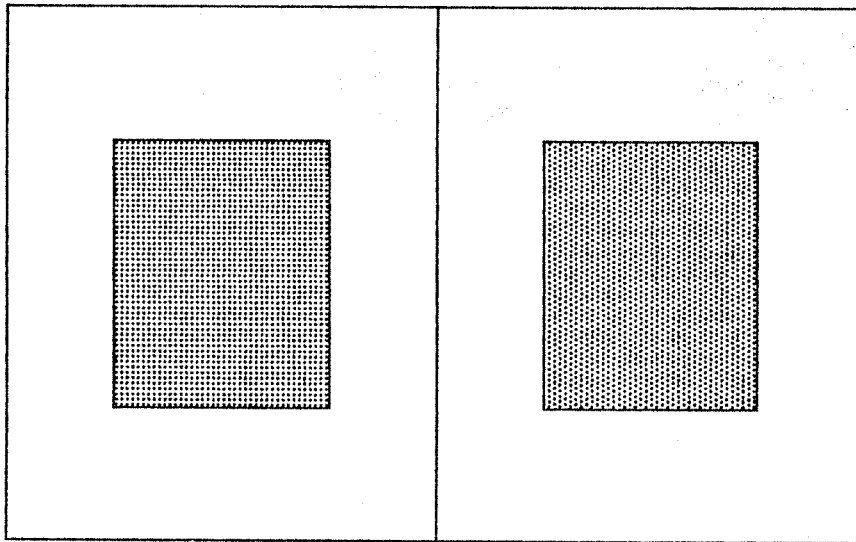


Fig. 4.13

En étudiant la grille diagonale, nous voyons que les lignes ayant des numéros pairs ont leurs points qui démarrent à la position zéro (la plus à gauche) tous les quatre pixels. Les lignes impaires ont leurs points qui démarrent à la position 2, également tous les quatre pixels. Supposez que le programme traite un bloc avec 3 pixels bleus et un rouge (avec le réglage COLOR 1,0). Dans les lignes paires, l'instruction

```
IF (ROW MOD 2=0 AND COLUMN MOD 4=0)
  THEN
    COL=2
  ELSE
    COL=0
```

règle COL à la bonne couleur. Dans les lignes impaires, les points rouges sont aussi placés, tous les quatre points, en commençant à la position 2. L'instruction

```
IF (ROW MOD 2<>0 AND COLUMN MOD 4=2)
  THEN
    COL=2
  ELSE
    COL=0
```

règle COL à la couleur correcte. En combinant les deux expressions nous avons

```

IF (ROW MOD 2=0 AND COLUMN MOD 4=0) OR
   (ROW MOD 2<>0 AND COLUMN MOD 4=2)
THEN
  COL=2
ELSE
  COL=0

```

Le programme 4.22 dessine deux rectangles et les remplit avec une combinaison qui a un point jaune pour chaque trois points bleus. Le rectangle de gauche est rempli en utilisant le coloriage horizontal et celui de droite est rempli en utilisant le coloriage diagonal, comme le montre la figure 4.13.

```

0  ** 4-22 : Horiz./diag. coloring **
10 SCREEN 1:COLOR 1,0:CLS
20 LINE(160,0)-(160,199),2
30 LINE(40,50)-(120,150),3,B:
   LINE(200,50)-(279,150),3,B
40 FOR X=41 TO 119
50   FOR Y=51 TO 149
60     IF X MOD 2=1 AND Y MOD 2=1
       THEN
         PSET(X,Y),3
       ELSE
         PSET(X,Y),0
70   NEXT
80 NEXT
90 FOR X=201 TO 278
100   FOR Y=51 TO 149
110     IF (Y MOD 2=0 AND X MOD 4=0)
       OR (Y MOD 2<>0 AND X MOD 4=2)
       THEN
         PSET(X,Y),3
       ELSE
         PSET(X,Y),0
120   NEXT
130 NEXT

```

PROGRAMME 4.22

4.14 COLORIAGE AVEC DES COULEURS NON STANDARDS

Nous présentons ici un algorithme pour colorier des formes relativement complexes avec n'importe quelle couleur de la palette étendue. Pour cela nous aurons besoin de la fonction POINT(X,Y) qui retourne le code couleur du point (X,Y).

La méthode est la suivante : étant donné un point (X,Y) et une couleur limitante (la couleur à laquelle le coloriage doit s'arrêter), le point à gauche du point (X,Y) est examiné avec POINT(X-1,Y). Si la couleur n'est pas la couleur limitante, le point est tracé et X est décrémenté de 1. L'étape est répétée jusqu'à ce qu'on trouve soit le bas de l'écran, soit

un point limitant. On applique alors la même procédure mais en se déplaçant cette fois sur la droite. Après ces deux étapes, la ligne sur laquelle se trouve (X,Y) a été coloriée. Y est décrémenté et la procédure est répétée jusqu'à ce qu'on arrive à ce qu'une ligne entière soit bloquée par la couleur limitante. Le programme 4.23 colore en utilisant un petit nombre de couleurs, mais en changeant le sous-programme de la ligne 1000, n'importe quelle autre couleur de la palette peut être obtenue.

```

0  '** 4-23 : Flooding, extended palette **
10 DEFINT A-Z: DIM C(5,2,2)
20 FOR K=1 TO 5:
    FOR I=0 TO 1:
        FOR J=0 TO 1:
            READ C(K,I,J):
            NEXT:
        NEXT:
    NEXT
30 SCREEN 1: COLOR 1,0: CLS
40 F=F+1: ON F GOTO 50,60,70,80,90
50  CIRCLE (200,100),20,3: X=200: Y=116:
    CO=2: GOSUB 1000: GOTO 40
60  CIRCLE (200,150),30,3: X=200: Y=174:
    CO=3: GOSUB 1000: GOTO 40
70  CIRCLE (100,150),30,3: X=100: Y=174:
    CO=4: GOSUB 1000: GOTO 40
80  CIRCLE (160,31),30,3: X=160: Y=55:
    CO=5: GOSUB 1000: GOTO 40
90 END
1000 'Flooding subroutine
1010 XT=X: X=X+1
1020 WHILE POINT(XT,Y)=0
1030  PSET(XT,Y),C(CO,XT MOD 2,Y MOD 2):
    XT=XT-1
1040 WEND:
    LEFT=XT+1
1050 WHILE POINT(X,Y)=0
1060  PSET(X,Y),C(CO,X MOD 2,Y MOD 2):
    X=X+1
1070 WEND: RIGHT=X-1
1080 Y=Y-1: X=LEFT:
    IF POINT(X,Y)=0
        THEN
            1010
        ELSE
            X=X+1
1090 WHILE POINT(X,Y)<>0 AND X<=RIGHT
1100  X=X+1
1110 WEND
1120 IF X=RIGHT+1
    THEN
        RETURN
    ELSE
        1010
5000 DATA 0,2,0,0,0,2,2,0,0,2,2,2,
          2,2,2,2,1,1,2,2

```

PROGRAMME 4.23

A la différence de PAINT, cet algorithme ne peut pas traiter des figures dans lesquelles il y a des embranchements (par exemple, si le départ n'est pas au bas de la figure, il laissera une partie de celle-ci non colorée). Si l'instruction pour colorier la forme laisse des parties non coloriées, il faut répéter le processus en partant de points situés à l'intérieur des zones non colorées.

4.15 LES COULEURS AVEC LES MONITEURS NOIR ET BLANC

On peut connecter un moniteur noir et blanc à la carte graphique. Tous les textes et les graphiques seront affichés. Le blanc sera blanc et le noir sera noir ³, mais les couleurs apparaîtront comme des mélanges de points noirs et blancs, rendant certains textes ou graphiques difficiles ou impossibles à déchiffrer. La table 4.1 montre à quoi ressemblent les textes et les graphiques avec les différents réglages de COLOR.

Table 4.1

<i>Palette</i>				
<i>0</i>			<i>1</i>	
<i>Fond</i>	<i>Graphiques</i>	<i>Texte</i>	<i>Graphiques</i>	<i>Texte</i>
0	TRES MAUVAIS	MAUVAIS	EXCELLENT	EXCELLENT
1	PASSABLE	TRES MAUVAIS	BON	BON
2	TRES MAUVAIS	TRES MAUVAIS	BON	BON
3	MAUVAIS	TRES MAUVAIS	BON	BON
4	MAUVAIS	TRES MAUVAIS	BON	BON
5	MAUVAIS	TRES MAUVAIS	BON	BON
6	TRES MAUVAIS	TRES MAUVAIS	BON	BON
7	PASSABLE	PASSABLE	TRES MAUVAIS	TRES MAUVAIS
8	BON	PASSABLE	EXCELLENT	EXCELLENT
9	PASSABLE	TRES MAUVAIS	BON	BON
10	TRES MAUVAIS	TRES MAUVAIS	BON	BON
11	MAUVAIS	TRES MAUVAIS	PASSABLE	PASSABLE
12	MAUVAIS	TRES MAUVAIS	BON	PASSABLE
13	PASSABLE	TRES MAUVAIS	BON	BON
14	TRES MAUVAIS	TRES MAUVAIS	BON	PASSABLE
15	EXCELLENT	EXCELLENT	PASSABLE	TRES MAUVAIS

CONDITIONS LIMITES

PAINT. Le point sur lequel le coloriage doit commencer doit être à l'intérieur de l'écran, ou une erreur « illegal function call » (appel d'une fonction non autorisée) sera induite.

Les couleurs de contour et de remplissage peuvent être n'importe quel nombre entre 0 et 255. Les valeurs fractionnaires sont arrondies, aussi les nombres supérieurs à -0,5 et inférieurs à 255,5 sont autorisés. Les valeurs supérieures à 3 produiront n'importe laquelle des quatre couleurs, en général la dernière tracée.

REVISION

PAINT(160,100),2,3

Remplit une zone limitée par la couleur 3 avec la couleur 2. Si le point (160,100) a déjà la couleur 3, l'instruction sera ignorée. Seule la couleur 3 fera s'arrêter le processus de coloriage, les autres couleurs seront également recoloriées.

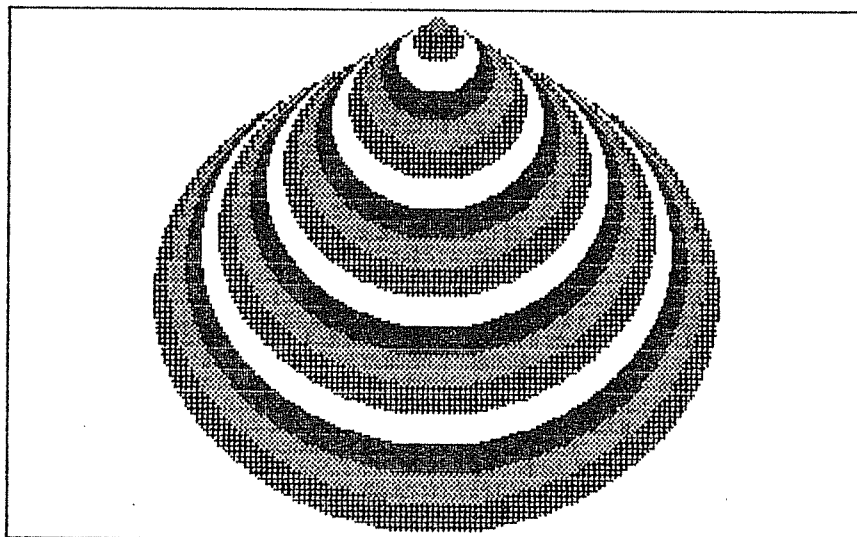


Fig. 4.14

EXERCICES

1. Ecrire un sous-programme avec une palette super étendue 4×4 .

2. Dans le tracé de lignes épaisses du programme 4.20, un grand nombre de blocs sont superposés. Si ces blocs sont dessinés avec une palette étendue, les calculs pour déterminer la couleur de certains points seront répétés plusieurs fois et le processus sera inexorablement long pour les grands blocs. Ecrire un programme dans lequel les points limites de certains des blocs précédents soient mémorisés et le calcul des couleurs minimisé.
3. Ecrire un programme pour dessiner un cercle (ou une ellipse) avec un trait épais.
4. Améliorer l'exercice 3 de manière à ce que le cercle puisse être dessiné avec une palette étendue et que la recherche des couleurs soit minimisée.
5. Ecrire un programme qui trace un polygone d'un nombre de côtés quelconque, mais avec un trait épais.
6. Ecrire un programme qui crée la figure 4.14. Notez qu'à la différence de la figure 3.2, les anneaux sont pleins.

NOTES

1. Si le centre du cercle est hors de l'écran, une erreur d'appel de fonction non autorisée sera induite. Ce cas est traité au paragraphe 4.4.
2. L'opérateur MOD ne peut être utilisé qu'avec des entiers (ceci dit, il ne peut manipuler des nombres supérieurs à 32767 ou inférieurs à -32768). Heureusement que, comme dans les graphiques, les coordonnées seront toujours inférieures à 320, ces limites sont plus qu'adéquates.
3. Les écrans en phosphore vert n'affichent pas du blanc mais du vert ; il existe également des écrans sur lesquels les parties brillantes apparaissent en fait orange.

5

La haute résolution

L'IBM a deux modes graphiques. L'écran 320 par 200 à quatre couleurs que nous avons utilisé dans les chapitres 1 à 4 est appelé le mode de moyenne résolution. Dans l'autre mode, de haute résolution, l'écran peut afficher 640 par 200 pixels en deux couleurs : noir et blanc. Presque toutes les instructions et fonctions se rapportant aux graphiques que nous avons présentés dans les chapitres précédents marchent également en haute résolution. Toutefois, à cause de la taille différente des pixels et de la limitation à deux couleurs, beaucoup d'arguments ont une signification légèrement différente. Dans ce chapitre nous expliquerons ces différences et nous présenterons quelques programmes bénéficiant de cette haute résolution.

5.1 PASSER EN HAUTE RESOLUTION

Il y a deux manières de régler un écran pour les graphiques haute résolution. La première est SCREEN 2. Si l'écran est déjà en haute résolution quand l'instruction est exécutée, elle n'a aucun effet, sinon l'écran est effacé et le LRP fixé à (320,100). Un autre effet est que les caractères affichés sont moitié moins larges que ceux affichés en moyenne résolution. Par conséquent, chaque ligne de texte a 80 caractères.

Si l'écran est en mode graphique moyenne résolution, une autre façon de passer en haute résolution sera « WIDTH 80 », qui a les mêmes effets que SCREEN 2. L'instruction WIDTH 40 fait passer de la haute à la moyenne résolution.

L'instruction CLS en haute résolution laisse le LRP à (320,100).

5.2 L'INSTRUCTION COLOR

L'instruction COLOR produit une erreur d'appel à une fonction non autorisée quand on l'utilise en haute résolution. En moyenne résolution on l'utilise pour fixer la couleur du fond et pour sélectionner la palette du premier plan. En haute résolution il n'y a qu'une couleur pour le fond et une couleur pour le premier plan, aussi cette instruction n'a aucun sens pour l'ordinateur.

5.3 PSET ET PRESET

En haute résolution les coordonnées de la colonne pour PSET peuvent aller de 0 à 639 et les points en dehors de cette gamme ne sont pas tracés du tout, au contraire de la moyenne résolution pour laquelle les valeurs des colonnes entre 320 et 639 sont tracées, mais à des positions erronées (voir paragraphe 1.18). Les couleurs 0 et 2 tracent les points en noir et les couleurs 1 et 3 tracent en blanc. Les autres valeurs positives inférieures à 255 sont autorisées mais ignorées. Les écrans couleur montrent tous les points avec la même intensité, mais certains écrans noir et blanc affichent des points isolés avec moins d'intensité que d'autres qui sont contigus à d'autres points. PRESET et PSET avec la couleur 0 se comportent de la même manière en haute résolution.

5.4 L'INSTRUCTION LINE

Outre les coordonnées étendues et les couleurs limitées, la seule différence pour l'instruction LINE entre la haute et la moyenne résolution est que les boîtes (rectangles pleins) avec des coordonnées en dehors de la gamme traçable ne sont pas affichées du tout en haute résolution. Ceci diffère de la moyenne résolution dans laquelle la gamme horizontale de 320 à 639 produit des boîtes erronées (voir paragraphe 2.7).

L'instruction LINE (0,0)-(639,199),1,BF peut être utilisée pour éclairer l'écran en blanc. Le LRP sera à (639,199).

5.5 L'INSTRUCTION CIRCLE

Afin de compenser la différence entre la taille horizontale et verticale des pixels en haute résolution, l'instruction CIRCLE est tracée avec un *aspect* de 5/12, à moins qu'on ne spécifie autre chose. Ceci permet d'utiliser de plus grands rayons que dans la moyenne résolution et le cercle

entier apparaissant toujours sur l'écran. L'instruction CIRCLE(160, 100),160, qui en moyenne résolution trace un cercle dont le haut et le bas n'apparaissent pas sur l'écran, tracera le cercle dans son entier en haute résolution.

5.6 LA PALETTE

Comme il n'y a que deux couleurs en haute résolution, il n'est possible d'étendre la palette qu'en des intensités de gris différentes. En utilisant des blocs 2×2 , il est possible d'avoir trois tons de gris. La figure 5.1 montre un des différents arrangements possibles.

Le programme 5.1 remplit cinq rectangles avec les couleurs de la palette étendue : noir, trois niveaux de gris et blanc. Le sous-programme de la ligne 1000 trace les points de coordonnées XS,YS avec la couleur correspondante à la position spécifiée par la variable COL. On peut utiliser ce sous-programme pour n'importe quelle autre forme. C'est l'équivalent de PSET pour la palette étendue.

```

0  '*** 5-1 : 2 * 2 block in hi-res **
10 SCREEN 2:CLS
20 DIM M%(3,1,1)
30 FOR COL=1 TO 3
40   FOR I=0 TO 1
50     FOR J=0 TO 1
60       READ M%(COL,I,J)
70     NEXT J
80   NEXT I
90 NEXT COL
100 FOR COL=1 TO 3
110   FOR X=0 TO 29
120     FOR Y=0 TO 29
130       XS=COL*30+X:YS=Y
135       GOSUB 1000
140     NEXT Y
150   NEXT X
160 NEXT COL
170 LINE(120,0)-(150,29),1,BF
180 FOR X=0 TO 120 STEP 30
190   LINE(X,0)-(X+29,29),1,B
200 NEXT X
210 W$=INPUT$(1):END
1000 'Plots point XS,YS with color C!
1010 PSET(XS,YS),M%(COL,XS MOD 2,YS I
1020 RETURN
10000 DATA 0,0,1,0
10010 DATA 0,1,1,0
10020 DATA 1,1,1,0

```

PROGRAMME 5.1

Un bloc 3×3 permet dix couleurs différentes. On peut voir une des distributions possibles des couleurs à la figure 4.11.

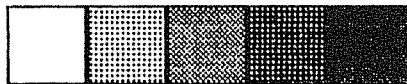


Fig. 5.1

Le programme 5.2 produit 10 rectangles avec les couleurs de la palette 3×3 : noir, huit niveaux de gris et blanc.

```

0  '** 5-2 : Ten colors in hi-res **
10 SCREEN 2:CLS
20 DIM M(10,2,2)
30 FOR COL=1 TO 10
40   FOR X=0 TO 2
50     FOR Y=0 TO 2
60       READ M(COL,X,Y)
70     NEXT
80   NEXT
90 NEXT
100 FOR COL=1 TO 10
110 FOR I=0 TO 31
120   FOR J=0 TO 40
130     PSET(I+(COL-1)*32,J),
        M(11-COL,I MOD 3,J MOD 3)
140   NEXT
150 NEXT
160 NEXT
170 FOR I=0 TO 300 STEP 32
180   LINE(I,0)-(I+31,41),1,B
190 NEXT
200 W$=INPUT$(1):END
5000 DATA 0,0,0,0,0,0,0,0,0,0
5010 DATA 0,0,0,0,0,1,0,0,0,0
5020 DATA 0,0,1,0,0,0,0,1,0,0
5030 DATA 1,0,0,0,0,0,1,0,1,0
5040 DATA 1,0,0,0,0,1,1,0,1,0
5050 DATA 1,0,1,0,1,1,0,1,0
5060 DATA 1,0,1,0,1,1,1,1,0
5070 DATA 1,0,1,1,1,1,1,1,0
5080 DATA 1,1,1,1,0,1,1,1,1
5090 DATA 1,1,1,1,1,1,1,1,1

```

PROGRAMME 5.2

5.7 LE TRACE DE FONCTIONS

Avec la haute résolution, il est possible d'atteindre un haut degré de précision. Les fonctions qui ont des petites variations sont plus faciles à visualiser quand leurs caractéristiques sont perceptibles grâce à la haute résolution. Une telle fonction est tracée par le programme 5.3, qui dessine plusieurs courbes sinusoïdales. La période de la fonction est graduellement réduite, et à la fin elle est si petite que les courbes se chevauchent presque.

```

0 '** 5-3 : Sine, decreasing frequency **
10 SCREEN 2:CLS:F=.01:DIM M(6)
20 FOR X=0 TO 639
30   Y=40*COS(X*F)
40   FOR J=2 TO 8
50     NY=5+19*J-Y:
       IF X>0
       THEN
         LINE(X-1,PY(J))-(X,NY)
60     PY(J)=NY
70   NEXT
80   F=F*1.003
90 NEXT
100 W$=INPUT$(1)

```

PROGRAMME 5.3

La figure 5.2 montre le graphique résultant.

5.8 LES EQUATIONS POLAIRES

Quand les courbes en équations polaires sont tracées très proches l'une de l'autre en moyenne résolution, l'image devient confuse. En général il est possible de faire se cotoyer un plus grand nombre de courbes dans le mode haute résolution. Il n'y a que trois changements nécessaires pour convertir en haute résolution les programmes écrits à l'origine en basse résolution :

L'instruction SCREEN 1 doit être changée en SCREEN 2.

Du fait de la largeur des pixels plus petite, l'échelle des X doit être doublée. On doit faire cela avant d'ajouter le décalage du centre.

Le centre de l'écran en haute résolution est le point (320,100), donc il faut ajouter le décalage des X en conséquence.

Le programme 5.4 est la version haute résolution du programme 3.23. La figure 5.3 montre le graphique produit quand $K = 11$.

```

0 '** 5-4 : High resolution stars **
10 SCREEN 2:CLS
20 PI=3.141595:R=99:DIM X(55),Y(55)
30 FOR K=1 TO 14:
   READ W:F=360/W
40   FOR I=0 TO W-1:A=I*F
50     X=R*COS(A*PI/180):Y=R*SIN(A*PI/180)
60     X(I)=2*X+320:Y(I)=Y+100
70   NEXT
80   FOR I=0 TO W-1
90     FOR J=I TO W-1
100      LINE(X(I),Y(I))-(X(J),Y(J))
110    NEXT
120  NEXT
130  W$=INPUT$(1):

```

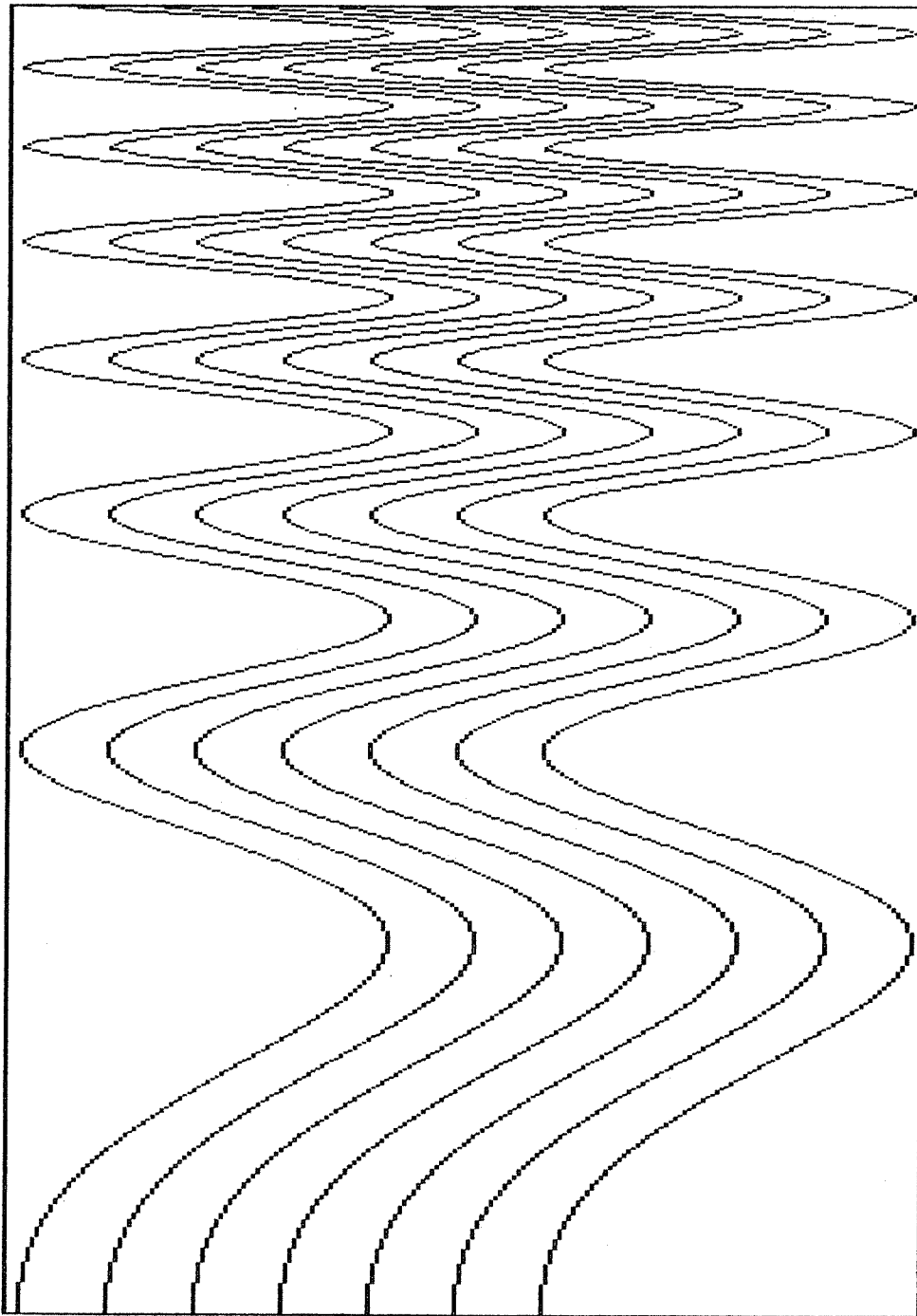


Fig. 5.2

```

      IF W$=CHR$(27)
      THEN
        END
      ELSE
        CLS
140 NEXT
5000 DATA 3,4,5,6,8,9,10,12,15,
          18,20,24,30,36

```

PROGRAMME 5.4

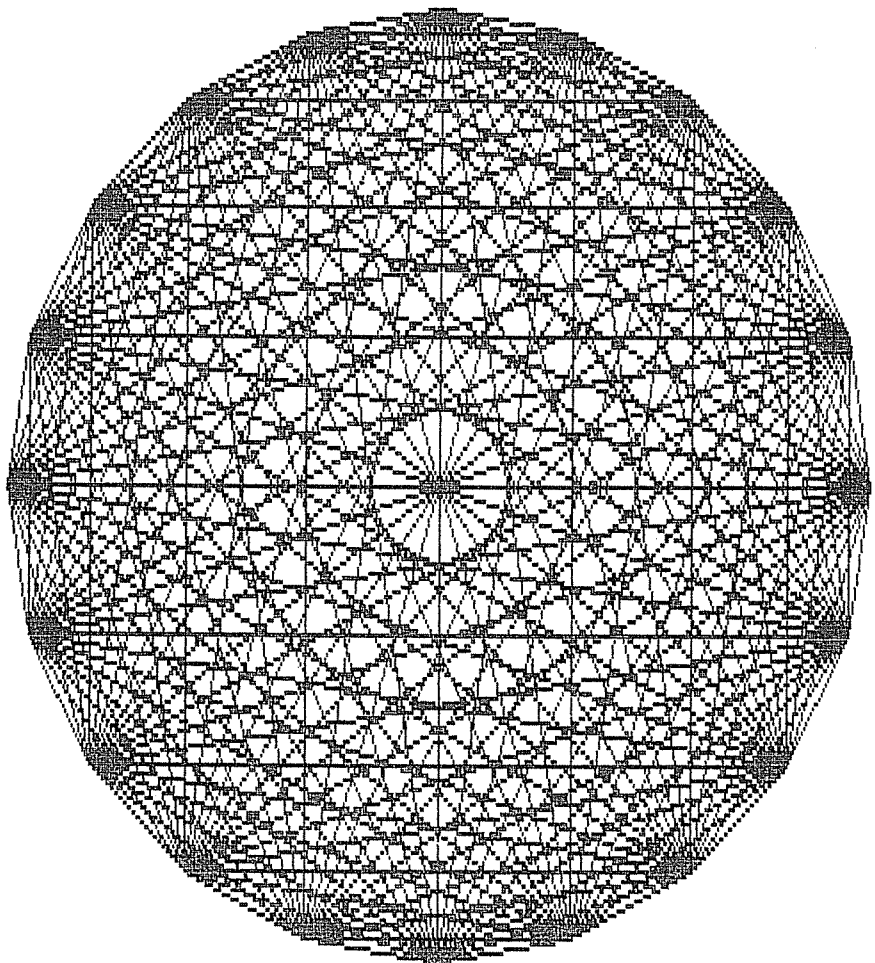


Fig. 5.3

Le programme 5.5 connecte les points de la périphérie d'un cercle au centre de l'écran. La figure 5.4 montre le graphique résultant.

```

0 '** 5-5 : Radial explosion **
10 F=1.745329E-02:R=99
20 SCREEN 2:CLS
30 FOR I=0 TO 360 STEP 1
40   X=320+2.2*R*COS(I*F)
50   Y=100+R*SIN(I*F)
60   LINE(320,100)-(X,Y)
70 NEXT
80 FOR I=0 TO 360 STEP 2
90   X=320+2.2*R*COS(I*F)
100  Y=100+R*SIN(I*F)
110  IF I=0
      THEN
        PSET(X,Y)
      ELSE
        LINE-(X,Y)
120 NEXT
130 PAINT(0,0),1,1

```

PROGRAMME 5.5

5.9 QUELQUES FIGURES INTERESSANTES

On peut réaliser un effet intéressant si des points blancs sont tracés sur un écran noir avec une concentration décroissante. Le programme 5.6 génère des nombres au hasard échelonnés horizontalement. Si le nombre généré est inférieur à 2, un point est tracé, et sinon rien. Sur la partie gauche de l'écran les nombres générés le seront dans un intervalle étroit, ainsi les nombres seront toujours inférieurs à 2 et les points affichés. Quand X croît, l'intervalle de définition des nombres aléatoires croît et les chances que les nombres soient inférieurs à 2 décroît. En conséquence, la concentration des points blancs décroît. La figure 5.5 montre le graphique produit.

```

0 '** 5-6 : From black to white **
10 SCREEN 2:CLS
20 FOR X=0 TO 639
30   FOR Y=0 TO 199
40     R=X*RND*.02
50     IF R<1
        THEN
          PSET(X,Y)
60   NEXT
70 NEXT

```

PROGRAMME 5.6

On peut utiliser une technique semblable pour colorer un cercle avec des points placés au hasard. Dans le programme 5.7, on utilise les valeurs de X et Y pour calculer la distance entre chaque point et le centre de l'écran. Si la distance est supérieure à 97, un point blanc est tracé. Ceci

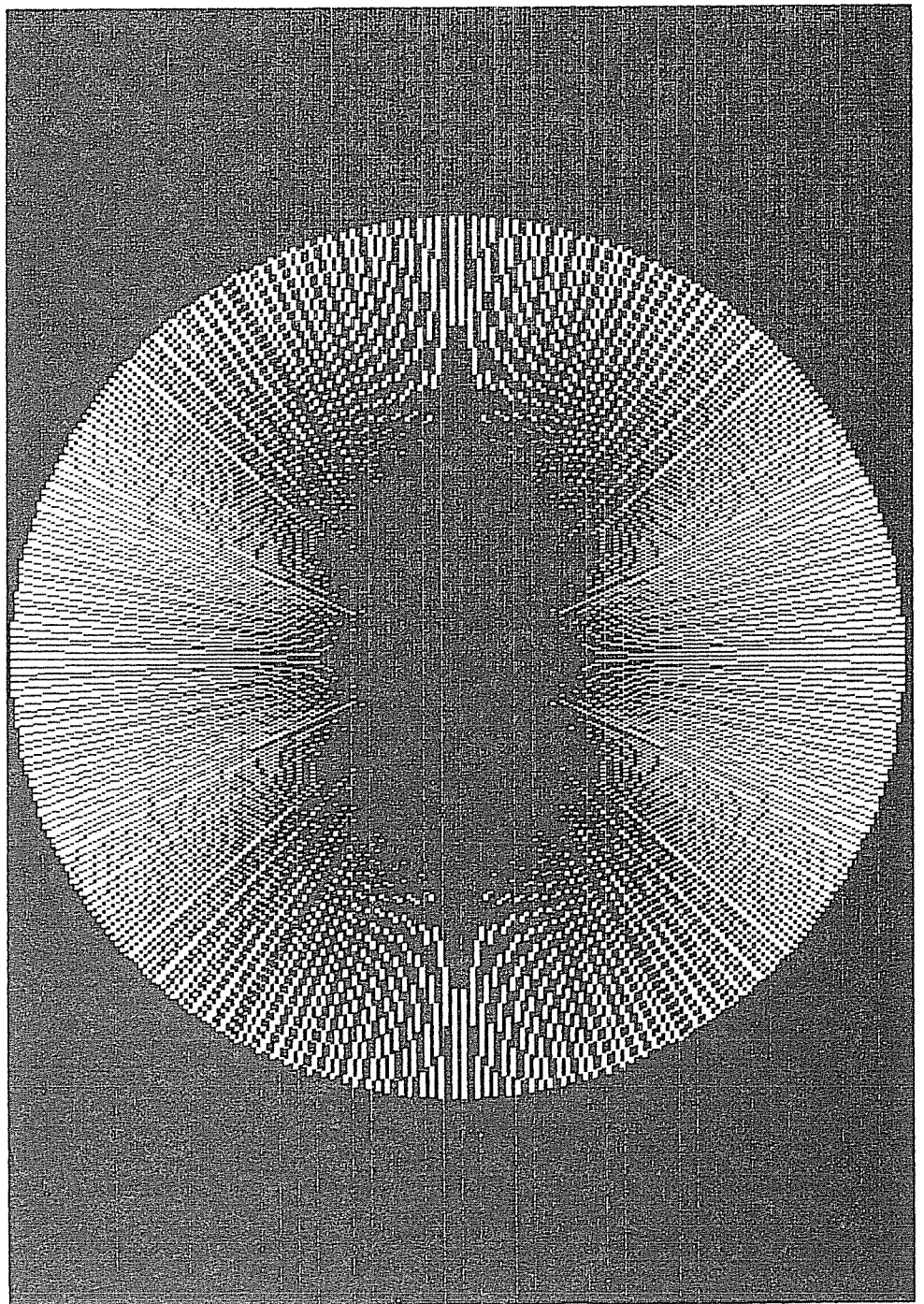


Fig. 5.4

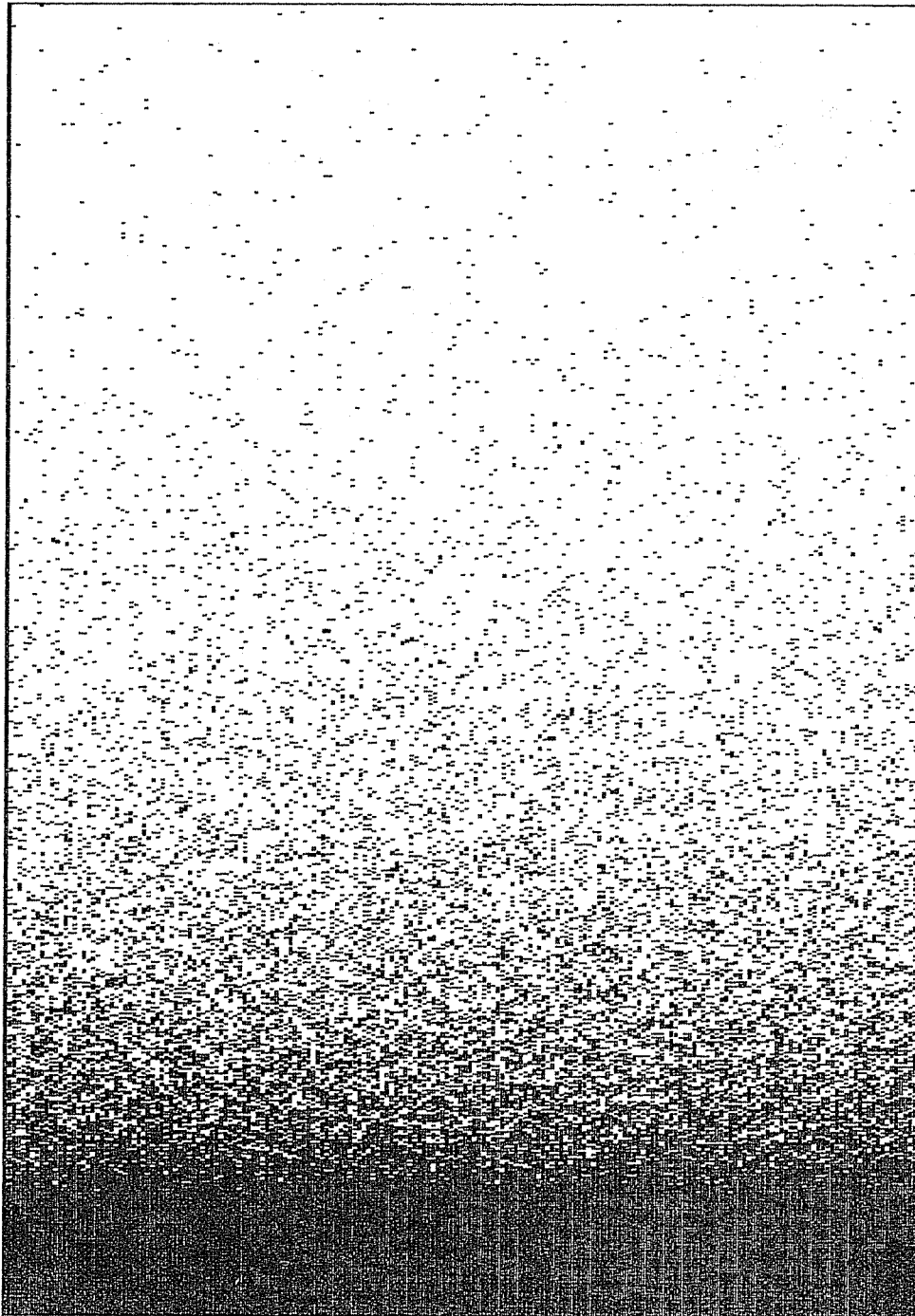


Fig. 5.5

remplit l'écran en blanc (noir sur la copie imprimée), sauf pour le cercle de rayon 97 centré en (320,100). Quand le point est à l'intérieur du cercle, un nombre aléatoire est généré, ses bornes dépendant de sa proximité à la périphérie du cercle. La figure 5.6 montre le graphique résultant.

```

0 '** 5-7 : Distant planet **
10 DEFINT X,Z
20 SCREEN 2:CLS:F=1.1
30 FOR X=0 TO 639
40   FOR Y=0 TO 199
50     DX=140-X*.44: DY=100-Y
60     D=SQR(DX*DX+DY*DY)
70     IF D>95
        THEN
          COL=1:GOTO 90
80     R=(95-D)^2*RND*.1:
        IF R<2
          THEN
            COL=1
          ELSE
            COL=0
90     PSET(X,Y),COL
100   NEXT
110 NEXT

```

PROGRAMME 5.7

Les programmes 5.8, 5.9 et 5.10 réalisent différentes nuances de gris en traçant des lignes rapprochées dans certaines parties de l'écran, éloignées dans d'autres. Les motifs qui apparaissent sont des franges d'interférence, dus aux effets de proximité de lignes possédant un nombre de points fini, à l'opposé de lignes continues formées d'une infinité de points élémentaires.

```

0 '** 5-8 : Vanishing plain **
10 SCREEN 2:CLS
20 FOR I=0 TO 639 STEP 5
30   LINE(0,0)-(I,199)
40 NEXT
50 FOR I=0 TO 199 STEP 3
60   LINE(0,0)-(639,I)
70 NEXT

```

PROGRAMME 5.8

```

0 '** 5-9 : Inside an eternal box **
10 SCREEN 2:CLS
20 FOR I=0 TO 639 STEP 5
30   LINE(320,100)-(I,0)
35   LINE(320,100)-(I,199)
40 NEXT
50 FOR I=0 TO 199 STEP 3
60   LINE(320,100)-(0,I)

```

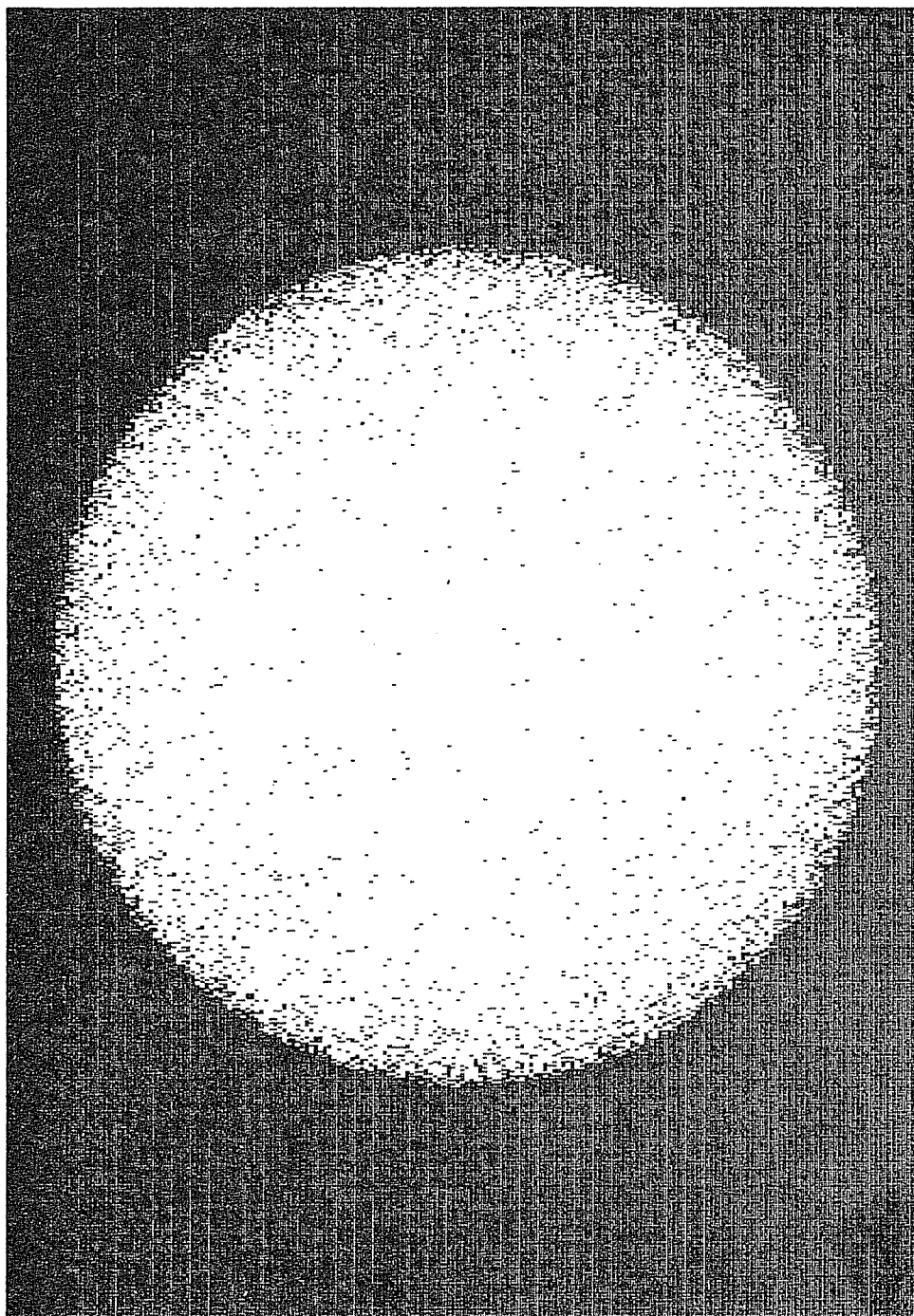


Fig. 5.6

```

65   LINE(320,100)-(639,I)
70 NEXT

```

PROGRAMME 5.9

```

0   ** 5-10 : Road to the sky **
10  SCREEN 2:CLS
20  FOR I=0 TO 639 STEP 5
30    LINE(320,0)-(I,199)
40  NEXT
50  FOR I=0 TO 319 STEP 5
60    LINE(0,199)-(I,0)
70    LINE(639,199)-(I+321,0)
80  NEXT

```

PROGRAMME 5.10

Les figures 5.7, 5.8 et 5.9 montrent les graphiques résultants. Vous pouvez faire des expériences en changeant le pas des boucles FOR dans les programmes précédents pour réaliser différents effets.

EXERCICES

1. Récrire le programme 2.8 de telle sorte que les limites de la fonction soient ajustées à l'écran haute résolution.
2. Modifier certains des programmes qui tracent des équations polaires (paragraphe 3.4.5) afin de générer un plus grand nombre de motifs complexes.
3. Changer le programme 5.3 de telle sorte que lorsque la période diminue, l'amplitude fasse de même.
4. Modifier le programme de l'exercice 3 de sorte que les lignes convergent sur le point (639,100).

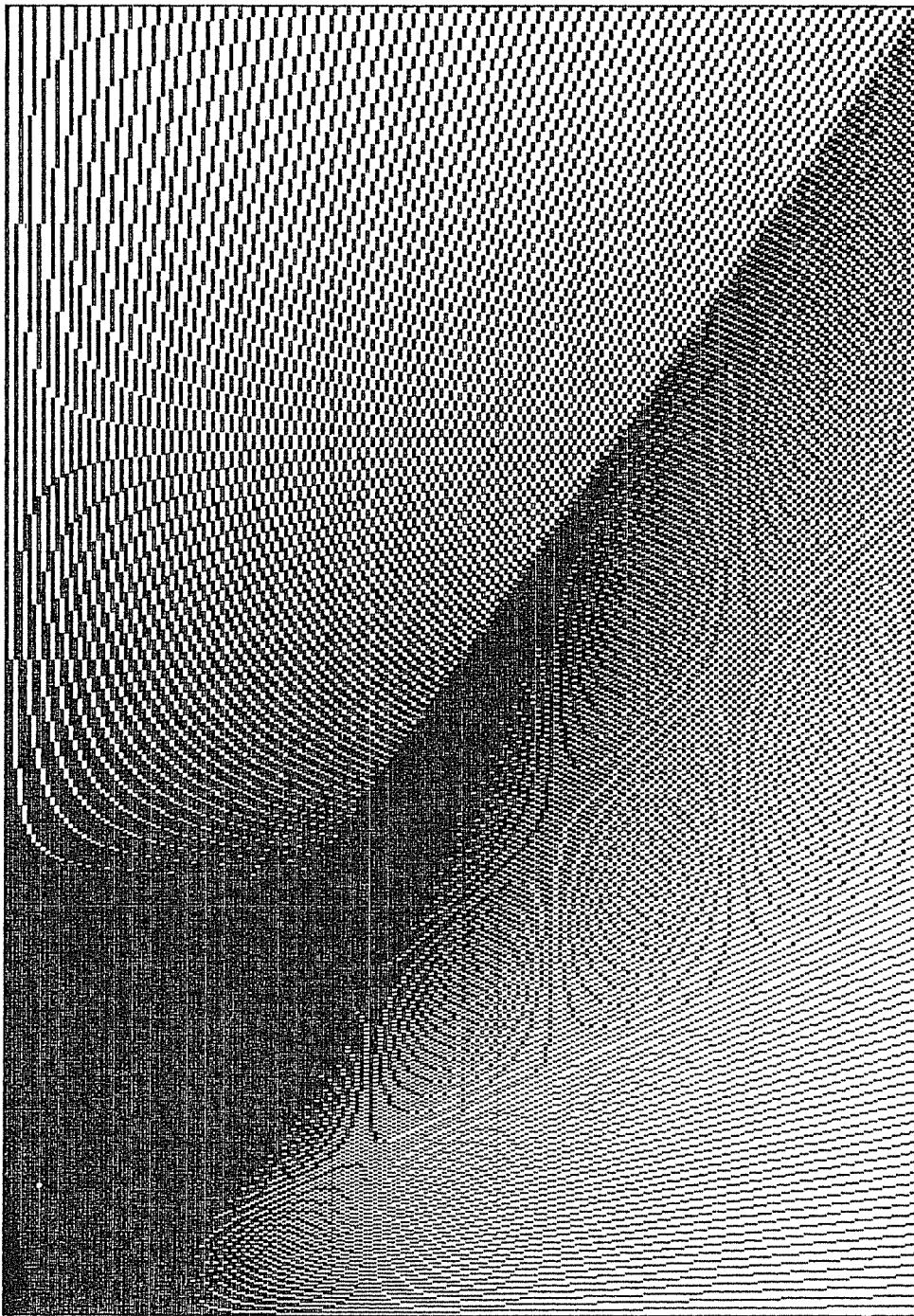


Fig. 5.7

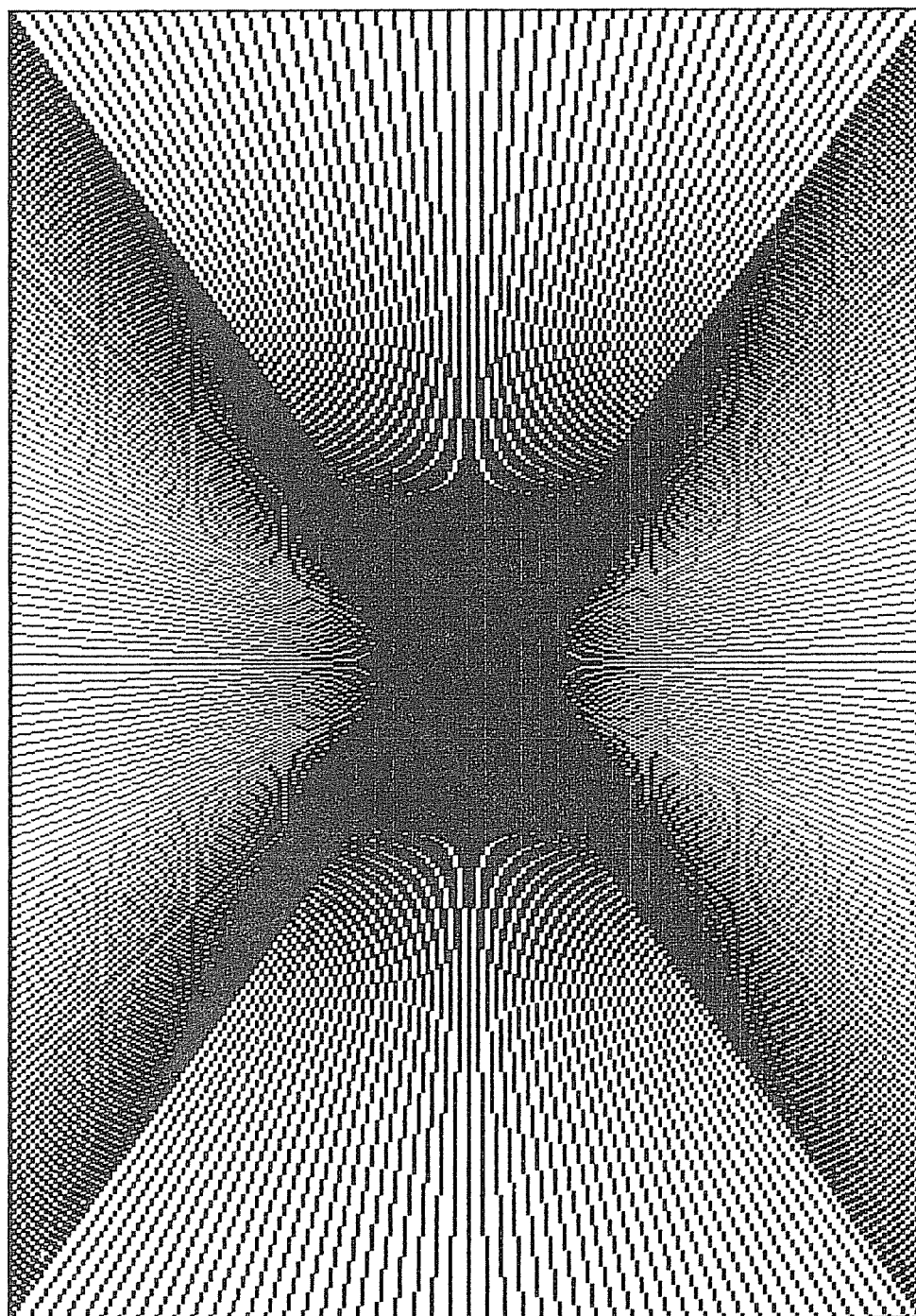


Fig. 5.8

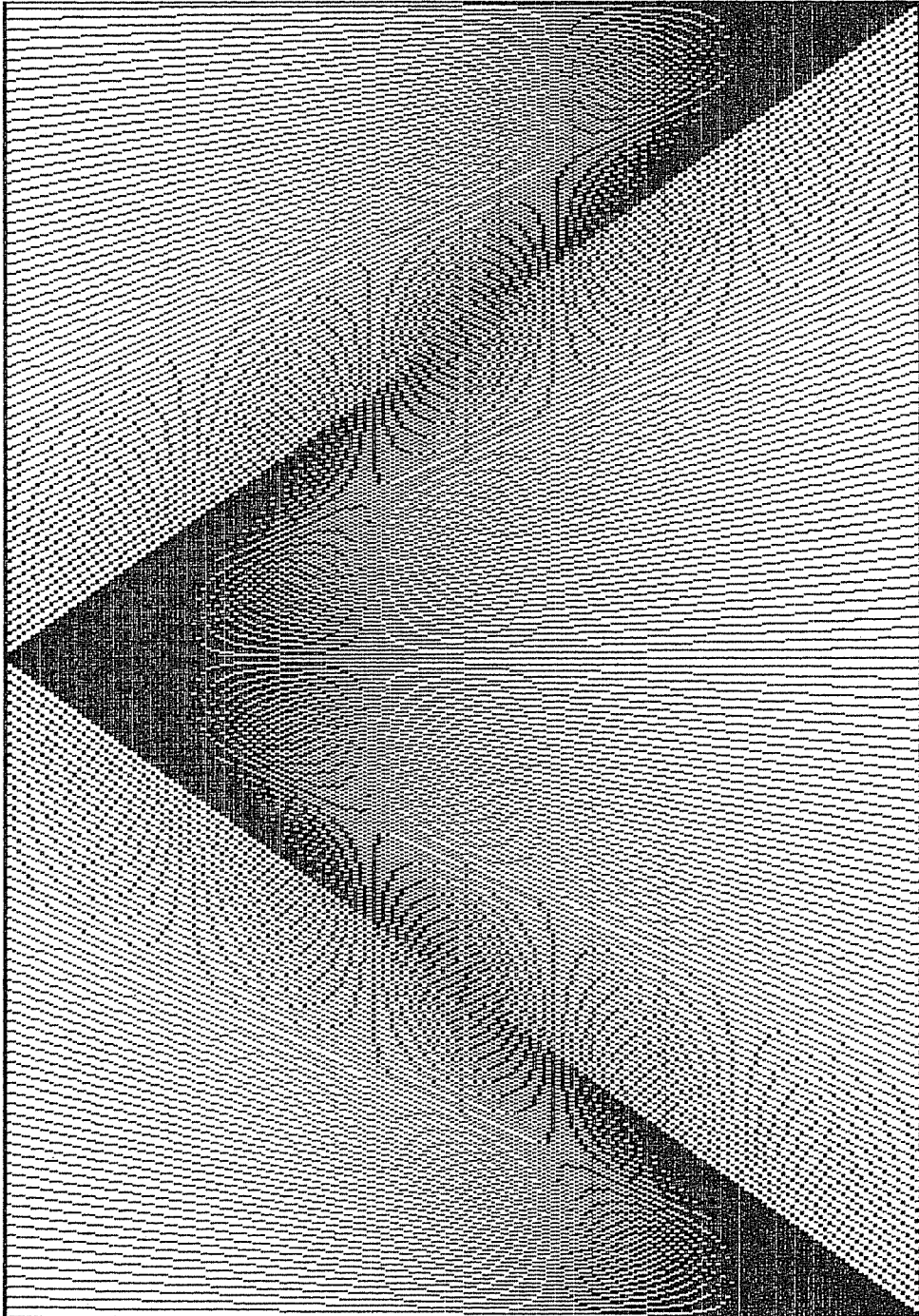


Fig. 5.9

6

Le sous-langage DRAW

L'instruction DRAW donne accès à tout un sous-langage spécialement destiné à définir des graphiques. Il existe des instructions pour tracer des lignes horizontales, verticales et diagonales, des instructions pour changer la couleur, des instructions pour agrandir, réduire ou faire tourner des figures, et des instructions pour déplacer le curseur. Ces instructions sont placées dans une chaîne de constantes ou de variables. Quand l'instruction DRAW est invoquée avec une telle chaîne, la figure désirée est dessinée.

La syntaxe de l'instruction est DRAW *chaîne* où *chaîne* contient les instructions que DRAW doit exécuter.

6.1 LIGNES HORIZONTALES ET VERTICALES

Il y a quatre instructions pour dessiner des lignes horizontales et verticales :

- U n Déplace de n unités vers le haut
- D n Déplace de n unités vers le bas
- L n Déplace de n unités sur la gauche
- R n Déplace de n unités sur la droite

Voici un exemple de l'utilisation de DRAW et des instructions que l'on vient de présenter : supposez que nous voulions dessiner un rectangle avec des côtés de 100 et de 30. Nous choisissons la trajectoire que l'on voit sur la figure 6.1. En utilisant les instructions U, D, L et R et les valeurs 100 et 30 nous créons la chaîne de dessin.

La chaîne résultante est `PICT$ = « R100 U30 L100 D30 »` qui peut être interprétée comme « se déplacer à droite de 100 unités, se déplacer vers le haut de 30 unités, se déplacer à gauche de 100 unités et finalement se déplacer vers le bas de 30 unités ». L'origine de la figure est le LRP. Les blancs dans la chaîne sont facultatifs mais aident à ce qu'elle soit plus lisible. Comparez ces deux formes « R100 U30 L100 D30 » et « R100U30L100D30 ».

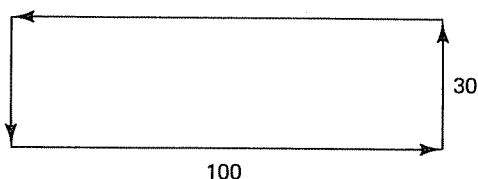


Fig. 6.1

Une fois que `PICT$` contient la chaîne d'instructions, la figure est tracée en exécutant simplement l'instruction `DRAW PICT$` et le rectangle apparaît directement sur l'écran. Une autre manière de produire le même résultat sera de placer directement les instructions dans une constante chaîne comme dans `DRAW « R100 U30 L100 D30 »`.

Chaque fois qu'en BASIC une valeur fractionnaire est introduite dans une instruction qui exige une valeur entière, la valeur fractionnaire est arrondie à l'entier le plus proche. Par exemple l'instruction `PSET(3.1, 100.8)` trace le point (3,101). Ceci n'est pas le cas pour les valeurs exigées dans les instructions DRAW. Dans les instructions DRAW, toutes les valeurs doivent être entières, et une valeur fractionnaire induira une erreur d'appel de fonction non autorisée.

Le LRP est toujours au dernier point dessiné par DRAW.

6.2 LIGNES DIAGONALES

Il existe quatre instructions pour tracer des lignes diagonales :

- E *n* Déplace diagonalement en haut et sur la droite de *n* unités.
- F *n* Déplace diagonalement en bas et sur la droite de *n* unités.

G *n* Déplace diagonalement en bas et sur la gauche de *n* unités.

H *n* Déplace diagonalement en haut et sur la gauche de *n* unités.

Dans les instructions de déplacement horizontal et vertical, la longueur de la ligne tracée, en pixels, correspond exactement à *n*. Dans les commandes de déplacement en diagonale, la longueur des lignes n'est pas vraiment *n*, mais *n* pixels horizontalement *n* verticalement. Par exemple, l'instruction DRAW « R50 U50 G50 » laissera le point courant à l'endroit où il se trouvait avant exécution de l'instruction, parce qu'il se déplace à droite de cinquante unités, en haut de cinquante, et ensuite diagonalement de cinquante unités à gauche et cinquante vers le bas. Le programme 6.1 dessine une maison en combinant des lignes horizontales, verticales et diagonales.

```
0  "" 6-1 : House ""
10 SCREEN 1:CLS
20 DRAW"U30 R30 H15 G15 R30 D30 L30"
```

PROGRAMME 6.1

Cette instruction supplémentaire ajoute une porte, comme le montre la figure 6.2.

```
30 DRAW"R5 U18 R10 D18".
```

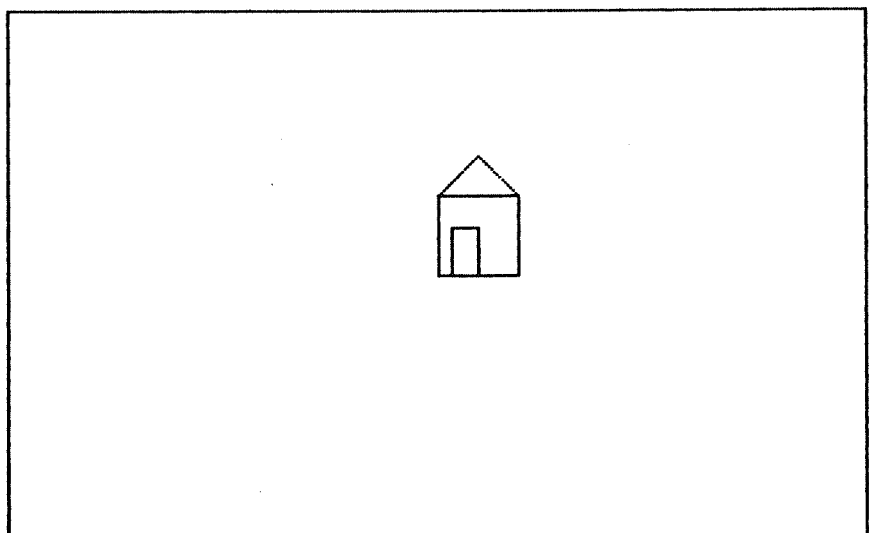


Fig. 6.2

6.3 POINTS EXTERIEURS A L'ECRAN

Quand une instruction DRAW essaie de tracer un point à l'extérieur de l'écran, la valeur la plus proche dans l'intervalle autorisé est utilisée. Si par exemple nous essayons de dessiner une maison, comme dans le programme 6.1, avec les valeurs multipliées par 3, le toit ne rentrerait pas dans l'écran et serait aplati de sorte que toute la figure soit dessinée dans l'intervalle des coordonnées autorisées comme le montre la figure 6.3.

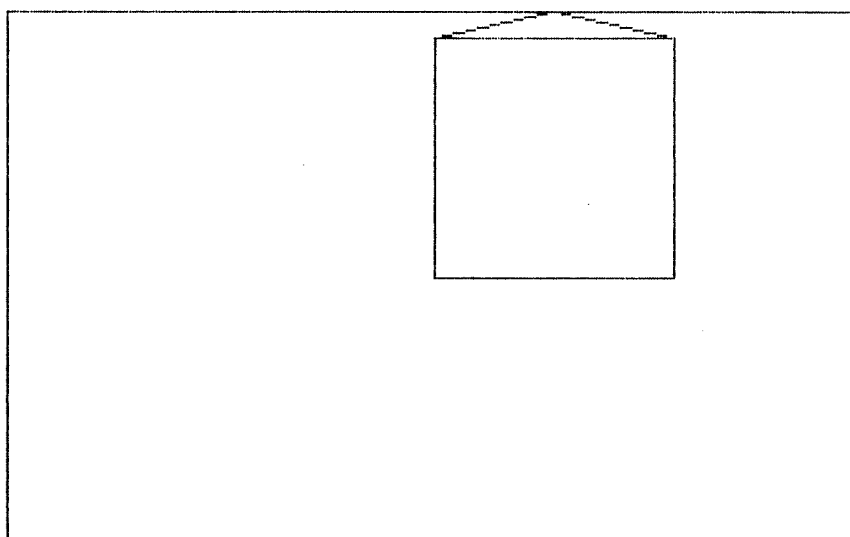


Fig. 6.3

La table 6.2 montre les points utilisés en réalité.

Table 6.2

INSTRUCTION	<i>Devrait laisser le point à</i>		<i>Laisse en fait le point à</i>	
	X	Y	X	Y
Point initial	160	100	160	100
U90	160	10	160	10
R90	250	10	250	10
H45	205	-35	205	0
G45	160	10	160	10
R90	250	10	250	10
D90	250	100	250	100
L90	160	100	160	100

Si les valeurs sont multipliées par quatre, le toit devient complètement plat et les murs sont raccourcis.

6.4 LE CHANGEMENT DE COULEUR

Il est possible de changer la couleur dans une chaîne de dessin en utilisant l'instruction C. Sa syntaxe est Cn, où n est 0, 1, 2 ou 3 pour la moyenne résolution et 0 ou 1 pour la haute résolution. La couleur prise par défaut est 3 pour la moyenne résolution et 1 pour la haute résolution. Le programme 6.2 dessine une maison avec un toit rouge et une porte verte.

```
0 "" 6-2 : Colored house ""
10 SCREEN 1:CLS:COLOR 0,0
20 DRAW""U30 R30 C2 H15 G15 C3 R30
    D30 L30 R5 C5 U18 R10 D18"
```

PROGRAMME 6.2

6.5 L'ECHELLE

L'instruction S détermine l'échelle du dessin : Sa syntaxe est Sn, où n divisé par quatre est le facteur d'échelle. n peut prendre des valeurs de 1 à 255. La valeur prise par défaut est 4, qui correspond à un dessin à l'échelle 1. Par exemple quand $n = 1$, le graphique est dessiné à l'échelle 1/4 par rapport aux paramètres de déplacements des instructions DRAW. Quand $n = 8$ la figure est dessinée à l'échelle 2.

L'instruction DRAW « R100 D40 », par exemple, trace une ligne 100 pixels sur la droite et 40 pixels en dessous du LRP. Une forme équivalente est DRAW « S4 R100 D40 » puisque $n/4 = 1$. Si on change le facteur d'échelle en 2, comme dans la ligne « S2 R100 D40 », une ligne 50 pixels sur la droite et 20 vers le bas sera tracée. La ligne DRAW « S1 R100 D40 » dessine une ligne 25 pixels sur la droite et 10 pixels vers le bas, et quand on fixe le facteur d'échelle à 8, comme dans la ligne DRAW « S8 R100 D40 » une ligne de 200 pixels sur la droite et 80 vers le bas sera dessinée.

Le programme 6.3 dessine une maison suivant quatre échelles différentes. L'instruction de la ligne 20 sert à laisser le LRP (0,199) parce que la plus grande maison ne rentrerait pas dans l'écran si on partait de la position réglée par CLS, qui est (160,100). La figure 6.4 montre les maisons produites.

```
0 "" 6-3 : Four houses ""
10 SCREEN 1:CLS
20 PSET (0,199)
30 DRAW""S1 U30 R30 H15 G15 R30 D30 L30"
40 DRAW""S4 U30 R30 H15 G15 R30 D30 L30"
```

```

50 DRAW"S8 U30 R30 H15 G15 R30 D30 L30"
60 DRAW"S16 U30 R30 H15 G15 R30 D30 L30"

```

PROGRAMME 6.3

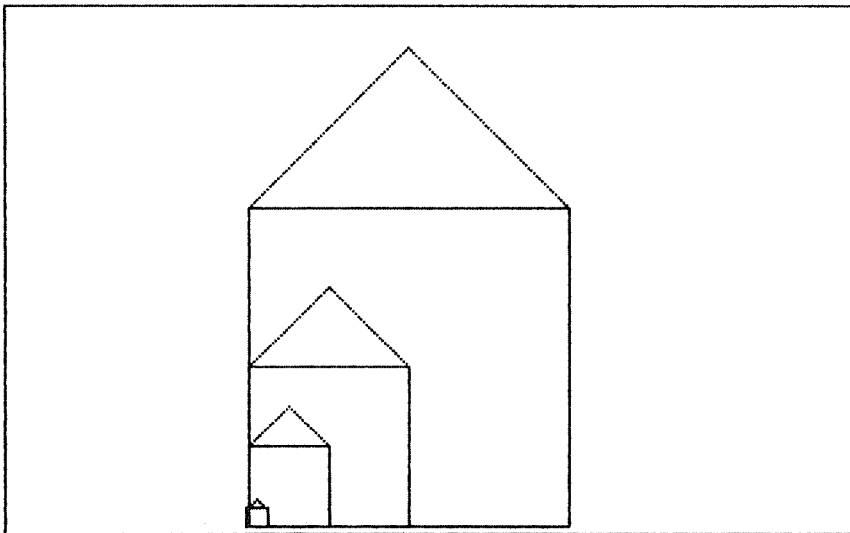


Fig. 6.4

On peut changer le facteur d'échelle n'importe où dans la chaîne de sorte que différentes parties peuvent être traitées suivant des échelles différentes.

6.6 L'UTILISATION DE VARIABLES COMME PARAMETRES

Jusqu'à maintenant, tous les paramètres de l'instruction DRAW ont été des constantes, ce qui rend le programme très rigide. On peut également utiliser des variables numériques en lieu et place des constantes, si toutefois les labels de ces variables sont précédés du signe égal. L'instruction DRAW « U50 R20 » est équivalente à I= 50:J= 20:DRAW « U= I;R= J; ».

Un point virgule est exigé immédiatement après le nom de la variable et on peut utiliser des variables simples ou en tableaux. La ligne T= 3: DRAW « U= T;R= T; » est équivalente à T(1,5)= 3:DRAW « U= T(1,5); R= T(1,5); » et aussi équivalente à X= 1:Y= 5:T(X,Y)= 3:DRAW « U= T(X,Y);R= T(X,Y); ».

On peut maintenant créer plus facilement une figure semblable à celle dessinée par le programme 6.3, comme le montre le programme 6.4.

```

0  '° 6-4 ; SCALING WITH VARIABLES °°
10 SCREEN 1:CLS:F=1
20 PSET (0,199)
30 FOR I=1 TO 10
40   DRAW"S=F;U30 R30 H15 G15 R30 D30 L30":F=F*2
50 NEXT

```

PROGRAMME 6.4

6.7 LES ROTATIONS

L'instruction A permet de sélectionner quatre orientations différentes. La syntaxe est An , où n peut varier de 0 à 3. n multiplié par 90 est l'angle de rotation appliqué. La figure 6.5 montre l'effet des quatre facteurs de rotation.

Si $n=0$, la rotation est de zéro degré (pas de rotation, ce qui est la valeur par défaut). Si $n=1$ la rotation est de 90 degrés ($\pi/2$ quand on mesure en radians) et ainsi de suite. La figure 6.6 montre la maison que l'on connaît bien dans les quatre rotations possibles.

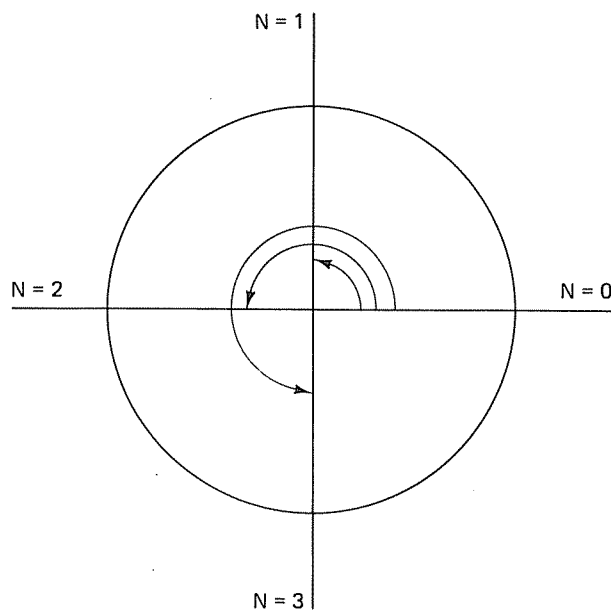


Fig. 6.5

Le programme 6.5 montre une flèche tournante. On utilise PICT\$ pour dessiner la flèche (couleur 3) et ERA.PICT\$ retrace le même chemin avec la couleur 0, l'effaçant. Quelquefois la rotation altère légèrement cer-

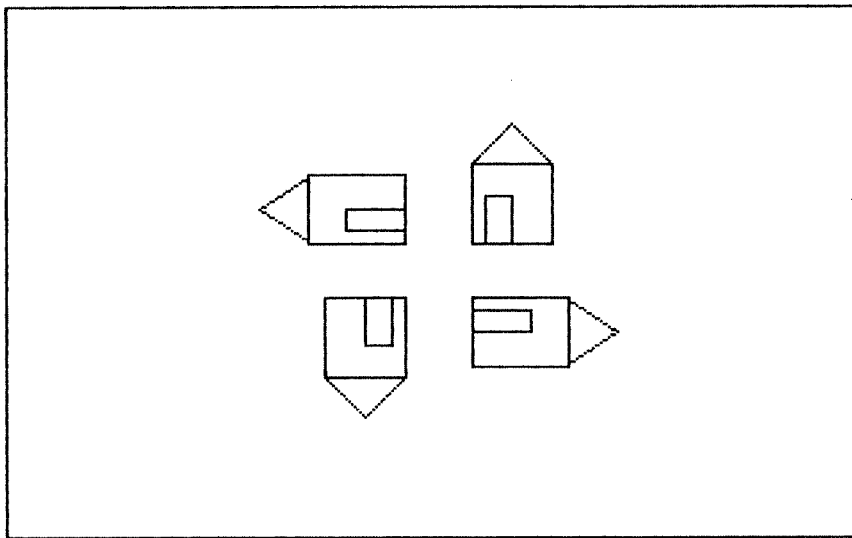


Fig. 6.6

tain proportions et le LRP peut ne pas correspondre exactement avec celui auquel on s'attendait. Dans ce cas même un seul pixel en moins ferait que la flèche ERA.PICT\$ manquerait sa cible (et ça le fait en fait), n'effaçant pas complètement le dessin. L'instruction PRESET de la ligne 50 laisse le curseur à la position (160,100).

```
0  "" 6-5 : Rotating arrow ""
10 SCREEN 1:CLS
20 PICT$="A=I;C3 U60 L20 D5 H10 E10 D5 R30 D70 L10"
30 ERA.PICT$="A=I;C0 U60 L20 D5 H10 E10 D5 R30 D70 L10"
40 FOR I=0 TO 3
50   PRESET (160,100)
60   DRAW PICT$
70   DRAW ERA.PICT$
80 NEXT:GOTO 40
```

PROGRAMME 6.5

6.8 L'UTILISATION DE CHAINES COMME SOUS-PROGRAMMES

De la même manière qu'on peut utiliser des variables numériques comme paramètres dans le sous-langage DRAW, on peut invoquer des variables chaînes de l'intérieur d'autres chaînes ou littéraux. La syntaxe est `X chaîne_variable;`. Dans le programme 6.5 nous devons utiliser deux chaînes successives pour dessiner et effacer une flèche tournante,

même si le libellé des deux chaînes était identique. C'était parce que nous n'avions pas de moyen direct de changer la couleur et reboucler sur la chaîne de dessin initiale. L'instruction X nous permet de n'utiliser qu'une seule chaîne, PICT\$ et de changer la couleur, comme le montre le programme 6.6. La ligne 60, par exemple, pourrait être interprétée comme « régler la couleur à 3 et GOSUB PICT\$ », ce qui est l'équivalent que de retrouver le contenu entier de PICT\$ après C3. Remarquez le point virgule après le nom de variable ; il faut toujours l'utiliser, même si la variable est le dernier article de la chaîne.

```
0  "" 6-6 : EXECUTING STRINGS ""
10 SCREEN 1:CLS
20 PICT$="A=I;U60 L20 D5 H10 E10 D5 R30 D70 L10"
40 FOR I=0 TO 3
50   PRESET (160,100)
60   DRAW"C3 XPICT$;"
70   DRAW"C0 XPICT$;"
80 NEXT:GOTO 40
```

PROGRAMME 6.6

De la même manière qu'un GOSUB peut se trouver à l'intérieur d'un autre sous-programme, il est possible d'appeler une variable de chaîne de l'intérieur d'une autre qui, à son tour, a été appelée par une autre, et ainsi de suite à l'aide de l'instruction X. La ligne 40 du programme 6.7 est équivalente à l'exécution de la ligne 10 du programme 6.8.

```
0  "" 6-7 : Nested strings ""
5  SCREEN 1:CLS
10 P1$="E10 F10"
20 P2$="U10 XP1$;XP1$;D10"
30 P3$="U10 XP2$;D10 XP2$; R10 XP1$;"
40 DRAW P3$
```

PROGRAMME 6.7

```
10 P3$="U10 U10 E10 F10 E10 F10 D10 D10 U10
    E10 F10 E10 F10 D10 R10 E10 F10"
```

PROGRAMME 6.8

6.9 LE DEPLACEMENT DU CURSEUR

Dans tous les exemples précédents des instructions DRAW, la première ligne commençait au LRP que quelquefois nous devons fixer en utilisant soit PSET soit PRESET. Il existe une instruction pour déplacer le curseur

des graphiques. Sa syntaxe est Mxy où x est la position de la colonne et y celle de la ligne. Le mouvement peut être soit absolu, soit relatif, mais ces modes sont indiqués d'une manière différente que dans les instructions graphiques standards. Des constantes non signées indiquent un mouvement absolu. Si la valeur x est précédée par un signe plus ou moins, le mouvement est relatif, et la valeur y peut être signée ou non (si elle n'est pas signée, elle est considérée comme positive).

L'action de la sous-instruction M est identique ¹ à celle de LINE-(X,Y). L'instruction DRAW « M 50,45 » est équivalente à LINE-(50,45), c'est-à-dire « tracer une ligne du LRP à (50,45) ». Pour utiliser des variables pour les valeurs x et/ou y , le format est $M = \text{variable};, = \text{variable};$. La ligne

```
100 PSET(160,100):I=25:DRAW"M-=I;,35"
```

déplace le curseur au point défini par $X = 135(160 - I = 160 - 25)$ et $Y = 135(100 + 35)$. Le programme 6.9 remplit l'écran avec des maisons. Avant de dessiner une nouvelle maison, la sous-instruction M place le curseur à la position correcte. La figure 6.7 montre le graphique résultant.

```
0  "" 6-9 : Moving the cursor ""
10 SCREEN 1:CLS
20 PICT$="U20 E10 F10 D20 L20"
30 FOR I=0 TO 300 STEP 29
49   FOR J=40 TO 199 STEP 35
50     DRAW"M=I;,=J; XPICT$;"
60   NEXT
70 NEXT
```

PROGRAMME 6.9

Chaque fois que le curseur se déplace, il trace son chemin, ce qui n'est pas toujours désirable. Dans le paragraphe suivant nous verrons comment déplacer le curseur sans dessiner.

6.10 DEPLACEMENT SANS TRACE

Quand n'importe laquelle des instructions de dessin est précédée par un B, l'instruction est exécutée comme si c'était avec une couleur invisible (pas la couleur du fond qui n'est pas vraiment invisible puisqu'elle efface tout sur son chemin). Le curseur est affecté exactement comme si l'ordinateur avait normalement obéi à l'instruction, mais aucun point n'est tracé. Une des utilisations de l'instruction B est de déplacer le curseur (avec l'instruction M) sans montrer la connexion de l'origine à la destination. Si on change la ligne 50 du programme 6.9 en :

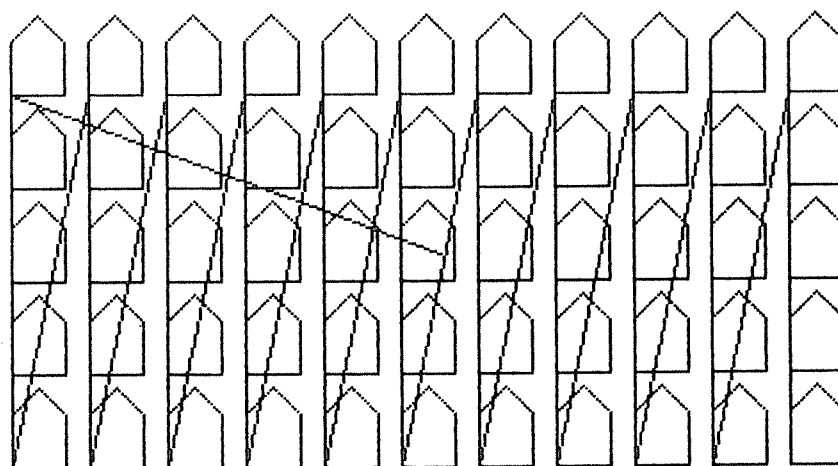


Fig. 6.7

```
50 DRAW "B M=I;=J; XPICT$;"
```

les maisons seront dessinées sans connexions entre elles, comme le montre la figure 6.8. Notez que le B n'affecte que l'instruction suivante, pas la ligne entière. Si nous changeons la ligne 20 du programme 6.9 en :

```
20 PICT$="B U20 E10 F10 B D20 L20"
```

les maisons seront dessinées avec des murs invisibles, comme le montre la figure 6.9.

Remarquez que sans la sous-instruction B, chaque fois que nous avons besoin de localiser le LRP à un endroit précis, nous utilisons PSET ou PRESET, aucune des deux méthodes n'étant sans inconvénient. Nous pouvons maintenant le relocaliser en toute sécurité. Pour déplacer le LRP au point (5,180), par exemple, utilisez simplement l'instruction DRAW « BM 5,180 ».

6.11 LE RETOUR DU CURSEUR A SON ORIGINE

Quand n'importe laquelle des instructions qui déplacent le curseur est précédée par l'instruction N, le LRP est renvoyé à sa position d'origine après que l'instruction ait été exécutée. Pour voir son effet, considérez la ligne suivante DRAW « R20 D20 L20 U20 ». Comme chaque instruction laisse le LRP au dernier point qu'elle trace, la figure résultante est un carré. Si nous faisons précéder chaque instruction DRAW par un N, comme dans DRAW « NR20 ND20 NL20 NU20 », le curseur sera

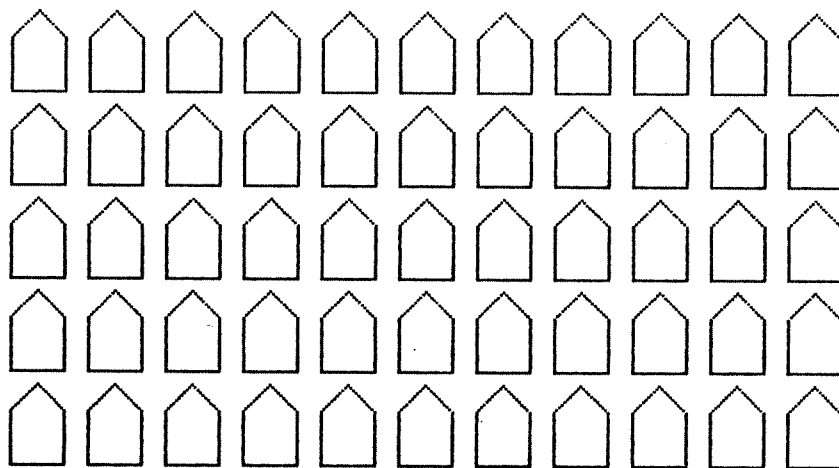


Fig. 6.8

renvoyé à sa position d'origine après que chaque ligne ait été dessinée et le résultat est la croix de la figure 6.10.

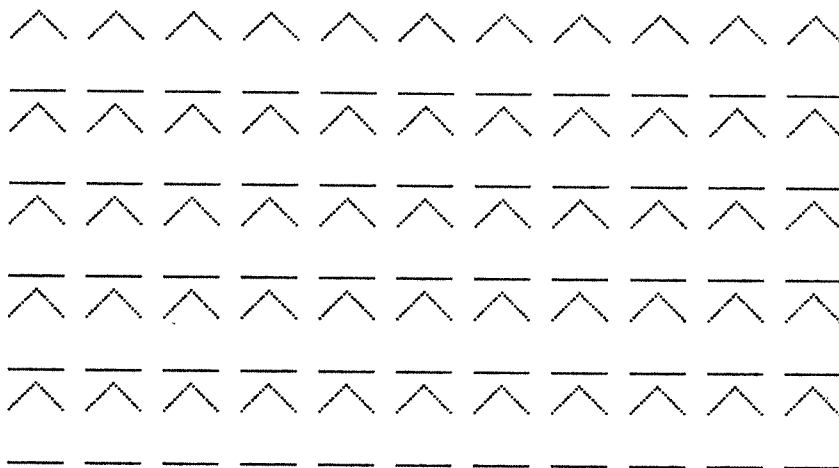


Fig. 6.9

6.12 DRAW COMPILE

Quand un programme va être compilé avec le compilateur BASCOM, la syntaxe des deux instructions peut être modifiée légèrement. Il existe deux moyens d'indiquer les variables dans une chaîne DRAW, comme le montre la table 6.3.

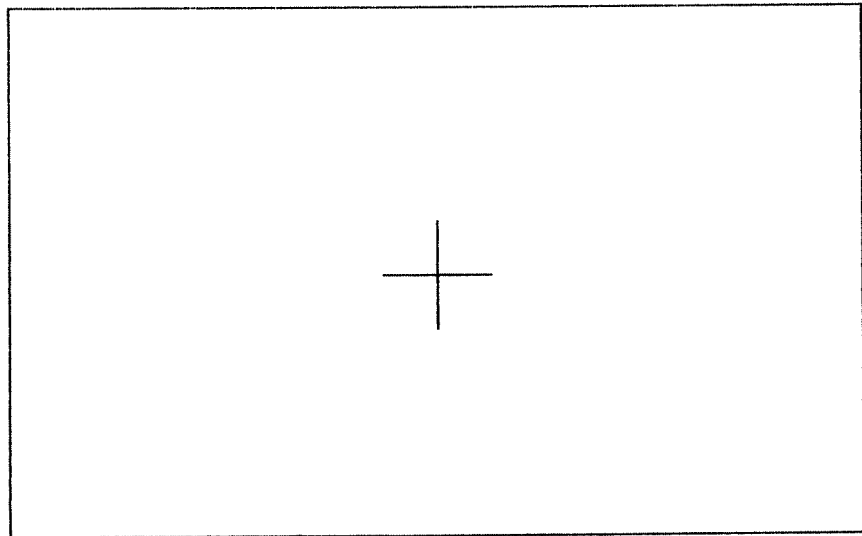


Fig. 6.10

Table 6.3

<i>SYNTAXE 1</i>	<i>SYNTAXE 2</i>
DRAW"U=DIST;"	DRAW"U=" + VARPTR\$(DIST)
DRAW"XPICT\$;"	DRAW"X" + VARPTR\$(PICT\$)

Les syntaxes 1 et 2 marchent bien en BASICA, mais SYNTAX 1 ne marche pas dans la version compilée.

REVISION

U <i>n</i>	Déplace vers le haut de <i>n</i> unités.
D <i>n</i>	Déplace vers le bas de <i>n</i> unités.
L <i>n</i>	Déplace sur la gauche de <i>n</i> unités.
R <i>n</i>	Déplace sur la droite de <i>n</i> unités.
E <i>n</i>	Déplace diagonalement en haut et à droite de <i>n</i> unités.
F <i>n</i>	Déplace diagonalement en bas et à droite de <i>n</i> unités.
G <i>n</i>	Déplace diagonalement en bas et à gauche de <i>n</i> unités.
H <i>n</i>	Déplace diagonalement en haut et à gauche de <i>n</i> unités.

C <i>n</i>	Règle la couleur à <i>n</i>
S <i>n</i>	Règle le facteur d'échelle à <i>n</i> et les mesures des instructions suivantes seront multipliées par <i>n</i> /4.
= <i>variables</i> ;	Utilise la valeur en cours de la <i>variable</i> dans l'instruction précédent immédiatement le signe égal.
A <i>n</i>	Règle la rotation à <i>n</i> . Ce <i>n</i> peut être 0, 1, 2 ou 3, et en le multipliant par 90 on a l'angle de rotation en degrés.
X <i>chaîne_variable</i> ;	Exécute <i>chaîne_variable</i> de l'intérieur d'une autre chaîne.
M <i>x,y</i>	Trace une ligne entre le LRP et (X,Y). Les coordonnées (X,Y) peuvent être sous forme absolue ou relative.
B	Exécute l'instruction DRAW suivante avec la couleur « invisible », ce qui revient à déplacer le curseur sans tracer de points.
N	Exécute l'instruction suivante mais ne change pas le LRP.
DRAW « U= » + VARPTR\$(DIST)	Equivalent à DRAW « U=DIST; »
DRAW « X » + VARPTR\$(PICT\$)	Equivalent à DRAW « XPICT\$; »

EXERCICES

1. Ecrire un programme qui génère la figure 6.6.
2. Ecrire un programme qui écrive « HI THERE » en utilisant les instructions DRAW.

NOTES

1. L'instruction M peut aussi être utilisée pour changer la position du LRP sans avoir d'effet notable sur l'écran comme cela est expliqué au paragraphe 6.10.

7

Le texte dans les graphiques

Tous les caractères standards¹ que l'on peut utiliser dans le mode texte peuvent être également affichés dans n'importe lequel des deux modes graphiques. Il y a 25 lignes de 40 caractères en moyenne résolution et 25 lignes de 80 caractères en haute résolution. La figure 7.1 montre les caractères qui peuvent être affichés sur l'écran graphique.

☐	☐	♥	♦	♣	♠	☐	☐	✱	▶	◀	↕
!!	¶	§	=	≡	↑	↓	→	←	?	"	#
\$	%	&	'	()	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;
<	=	>	?	@	A	B	C	D	E	F	G
H	I	J	K	L	M	N	O	P	Q	R	S
T	U	V	W	X	Y	Z	[\]	^	_
`	a	b	c	d	e	f	g	h	i	j	k
l	m	n	o	p	q	r	s	t	u	v	w
x	y	z	{		}	~	Δ				

Fig. 7.1

7.1 LES CURSEURS

Dans tous les chapitres précédents, chaque fois que nous avons mentionné le mot curseur, nous faisons allusion au LRP. Il faut faire ici une distinction entre le curseur de texte et le curseur de graphiques. Dans un environnement de texte le mot curseur indique le carré (ou trait de repère) qui apparaît avec chaque instruction INPUT. Ce carré sert à indiquer la destination du prochain caractère tapé. Comme ce type de curseur n'apparaît normalement qu'avec les instructions INPUT, on utilise le même mot (curseur) pour indiquer la position où le nouveau caractère envoyé à l'écran apparaîtra, notion similaire à celle du LRP, que nous pourrions appeler le dernier caractère envisagé dans le programme. Comme ce chapitre traite des textes, nous utiliserons le terme curseur pour indiquer à la fois le carré (curseur physique) et le dernier caractère envisagé (curseur logique). Chaque fois que le terme pourrait être ambigu, nous précisons auquel des deux nous nous référons. Le curseur de graphiques sera toujours appelé le LRP (dernier point envisagé).

7.2 POSITIONS DES CARACTERES

Les caractères affichés sur l'écran par les méthodes proposées dans ce but par BASIC (PRINT, WRITE, PRINT#, WRITE#) peuvent seulement être placés sur une des 25 lignes de 40 ou 80 caractères. Chaque caractère est positionné dans un carré de huit × huit pixels, dont on n'utilise en fait qu'une zone de 6 × 7 pixels, laissant ainsi une séparation horizontale de deux pixels pour deux caractères contigus. Peu de lettres descendent jusqu'au huitième pixel (j, p, q, y) et certains caractères spéciaux comme « \$ » occupent pratiquement la totalité de la cellule.

Le programme 7.1 génère des boîtes autour de toutes les cellules de caractères. La figure 7.2 montre la grille créée, quelques caractères y étant placés au hasard.

```
0  "" 7-1 : Character cells ""
10 SCREEN 1:COLOR 0,0:CLS
20 FOR X=0 TO 319 STEP 8
30   LINE(X,0)-(X,199),2
40   LINE(X+7,0)-(X+7,199),2
50 NEXT
60 FOR Y=0 TO 199 STEP 8
70   LINE(0,Y)-(319,Y),2
80   LINE(0,Y+7)-(319,Y+7),2
90 NEXT
```

PROGRAMME 7.1

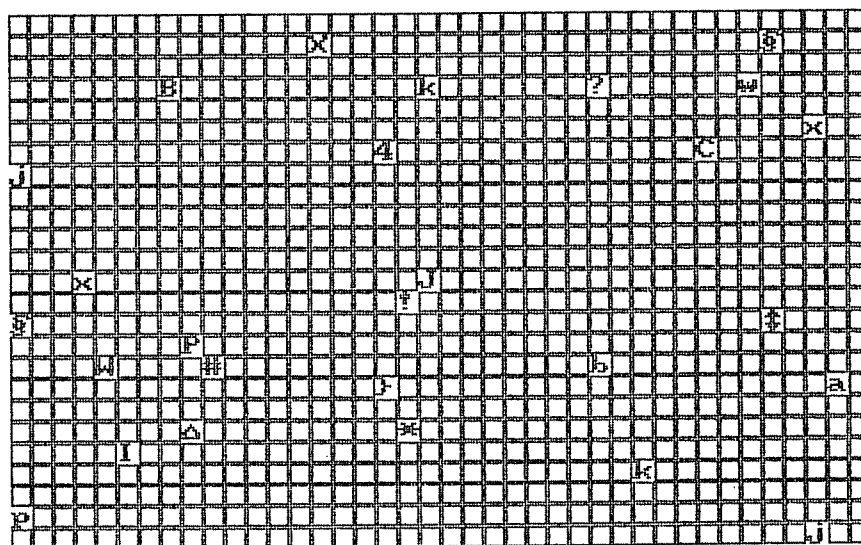


Fig. 7.2

Pour que le programme 7.1 produise une grille en haute résolution, changez simplement les 319 en 639, la couleur 2 en 1 et SCREEN 1 en SCREEN 2.

La position du caractère est indiquée par un couple de valeurs (X,Y), où X est le numéro de colonne et Y le numéro de ligne. Les colonnes vont de 1 à 40 en moyenne résolution et de 1 à 80 en haute résolution. Les lignes vont de 1 à 25. En termes de coordonnées graphiques, le coin en haut à gauche de chaque cellule de caractère est localisé par le point

$$((X-1)*8,(Y-1)*8) \quad (7.1)$$

Inversement, les colonne et ligne de la cellule de texte affectée par un point graphique de coordonnées (X,Y) sont ²

$$(X\backslash 8+1,Y\backslash 8+1) \quad (7.2)$$

Exemple 1.

On peut maintenant dessiner la grille de texte d'après les positions de caractères, en faisant la conversion d'emplacement de caractère à emplacement graphique selon la formule 7.1. On peut utiliser deux boucles FOR pour tracer chaque ligne et colonne de caractères comme le montre le programme 7.2.

```
0  '°° 7-2 : Cells calculated °°
10 SCREEN 1:CLS
```



```

20 FOR I=1 TO 40
30   FOR J=1 TO 25
40     X=(I-1)*8:Y=(J-1)*8
50     LINE(X,Y)-(X+7,Y+7),2,B
60   NEXT J
70 NEXT I

```

PROGRAMME 7.2

Exemple 2.

Supposez que les trois lignes du haut de l'écran affichent le texte que l'on voit à la figure 7.3. Nous voulons savoir quel caractère sera affecté par la commande PSET(51,19). A l'aide de la formule 7.2, nous pouvons trouver que la position du caractère est

$$X = 51 \setminus 8 + 1 = 7$$

$$Y = 19 \setminus 8 + 1 = 3$$

Donc, le caractère affecté est situé à (7,3), c'est le « e » de « time ».

7.3 POSITIONNEMENT DU CURSEUR

CLS laisse le curseur à la position (1,1), le coin en haut à gauche de l'écran. Le premier caractère imprimé sera placé là et le curseur sera envoyé à la position (2,1). Chaque nouveau caractère envoyé à l'écran sera affiché à la position suivante sur la droite. Quand il n'y a plus assez de place pour placer le caractère ou le nombre envoyé à l'écran, le curseur est positionné dans la première colonne de la ligne inférieure suivante, et l'impression se poursuit à partir de là.

Il existe une instruction pour positionner le curseur à un emplacement précis et supplanter le positionnement séquentiel automatique. Son format est LOCATE Y,X où X est le numéro de colonne et Y le numéro de ligne³.

```

┌ If I had only known
└ This is the time
  No time at all

```

Fig. 7.3

7.4 LE DEROULEMENT DE L'ECRAN

Quand un caractère est imprimé sur la position la plus à droite de la ligne 24, l'écran est automatiquement « déroulé », c'est-à-dire que la ligne de texte un est remplacée par la ligne deux, la ligne deux par la ligne trois, etc..., pour laisser la ligne 24 vide et prête à recevoir le caractère suivant. Ce mouvement de caractères déplace tous les graphiques sur l'écran (tout est déplacé vers le haut de huit pixels, sauf la ligne 25).

La ligne 25 est normalement utilisée pour afficher les cinq premiers caractères des touches de fonctions. En moyenne résolution, seules les cinq premières clés sont affichées, alors qu'en haute résolution toutes vont sur l'écran. Lorsqu'on affiche les contenus des touches de fonctions, toute tentative de placer le curseur sur la ligne 25 avec l'instruction `LOCATE 25,X` provoque une erreur d'appel de fonction non autorisé. L'instruction `KEY OFF` enlève l'affichage des touches de fonctions et permet d'utiliser la ligne 25 comme ligne de texte normale, avec toutefois quelques limites.

L'instruction `KEY ON` restaure l'affichage de touches de fonctions.

Après l'instruction `LOCATE 25,X`, la ligne 25 peut être utilisée comme n'importe quelle autre ligne pour afficher des caractères. Toutefois quand le caractère le plus à droite est imprimé (le 40ème ou le 80ème) l'écran est déroulé à partir de la 24ème ligne et au-dessus. Le contenu de la ligne 25 n'est jamais déroulé et peut seulement être changé par un nouveau `PRINT`, un `CLS`, une instruction `KEY ON` ou par des instructions graphiques.

`INPUT` a quelques particularités quand on le fait sur la 25ème ligne. Si tous les caractères tapés rentrent dans la ligne, la rentrée se passe normalement, si non seuls les caractères qui sont à l'extérieur de la 25ème ligne sont en fait reçus par la variable d'entrée.

L'accès à la 25ème ligne ne peut se faire qu'avec l'instruction `LOCATE 25,X`, jamais avec une instruction `PRINT` normale.

7.5 LA COULEUR DES CARACTERES

Les caractères sont normalement imprimés dans la couleur par défaut pour le premier plan (numéro 3). En moyenne résolution, il est possible de changer cette couleur en exécutant `DEF SEG:POKE &H4E,couleur`, où *couleur* est une valeur entre 1 et 3. La couleur du fond (couleur numéro 0) n'est pas autorisée, et par conséquent cette méthode ne peut pas être utilisée pour afficher des caractères noirs sur fond blanc. En haute résolution, l'instruction n'a aucun effet.

Le programme 7.3 affiche des caractères dans les trois couleurs de premier plan.

```

0  "" 7-3 : Colored characters ""
10 SCREEN 1:COLOR 0,0:DEF SEG:CLS
20 POKE &H4E,1:LINE INPUT W$
30 POKE &H4E,2:LINE INPUT W$
40 POKE &H4E,3:LINE INPUT W$

```

PROGRAMME 7.3

Dans le paragraphe 7.8 nous présenterons une méthode pour afficher des caractères de n'importe quelle couleur sur n'importe quel fond.

7.6 LE REPOSITIONNEMENT DES CARACTERES

Nous allons maintenant vous présenter une méthode pour déplacer des caractères à des positions qui ne sont pas celles de caractères. Une technique plus rapide pour arriver au même résultat est présentée aux paragraphes 9.8, 9.9 et 9.10 mais la méthode que nous présentons ici permet un meilleur contrôle et prouvera qu'elle est essentielle dans les paragraphes à venir.

La fonction POINT renvoie le code couleur du pixel dont les coordonnées sont données comme arguments. Par exemple l'instruction COL=POINT (35,120) règle COL au code couleur du point (35,120). De cette façon, déplacer un ou plusieurs caractères devient peu de choses. Les étapes à suivre sont :

Imprimer le texte à partir d'une position connue ((1,1) est un bon choix du fait qu'il économise des calculs supplémentaires).

Calculer la zone couverte par le texte, à l'aide de la formule 7.1.

Choisir un point comme destination du texte (le coin en haut à gauche de la « fenêtre de destination »).

Utiliser deux boucles FOR et la fonction POINT pour balayer la zone de texte, en recherchant les points de couleur 3. Chaque fois qu'on en trouve un, faire PSET sur un point démarrant au point de destination plus le décalage des variables des boucles FOR. Les pixels aux couleurs différentes de 3 sont copiés avec PRESET.

Le programme 7.4 illustre cette méthode.

```

0  "" 7-4 : Relocates characters ""
10 SCREEN 1:COLOR 0,0:CLS
20 DESTX=50:DESTY=100
30 LINE INPUT W$
40 FOR X=0 TO 159
50   FOR Y=0 TO 7
60     IF POINT(X,Y)=3

```

```

        THEN
            PSET(DESTX+X,DESTY+Y)
        ELSE
            PRESET(DESTX+X,DESTY+Y)
70    NEXT
80    NEXT

```

PROGRAMME 7.4

7.7 MELANGE DES CARACTERES AVEC LA COULEUR DU FOND

Lorsqu'on place un caractère sur l'écran par une instruction PRINT ou similaire, les parties de la cellule du caractère ayant une couleur différente de 3 sont inscrites dans la couleur du fond. Le processus peut être vu comme : d'abord éclairer la totalité de la cellule à la couleur 0, et ensuite imprimer le caractère.

Pour imprimer les caractères sans toucher à la couleur du fond (sauf bien sûr pour la partie couverte par le caractère), changez simplement la ligne 60 du programme 7.4 en :

```

60    IF POINT(X,Y)=3
        THEN
            PSET(DESTX+X,DESTY+Y)

```

Ainsi, les points du caractère qui sont allumés seront tracés, mais le reste de la cellule ne sera pas touché du tout. Le programme 7.5 remplit une zone de l'écran avec des lignes dans la couleur 3. Les boucles FOR des lignes 60 à 95 copient chaque point des 13 premiers caractères tapés. Si le point a la couleur 3, il est copié à deux endroits, sinon il est seulement copié au-dessus. La figure 7.4 montre la figure résultante. Du fait que les lignes et les caractères sont tous les deux noirs sur la copie noir et blanc, le résultat est confus. Toutefois, c'est tout à fait clair sur l'écran couleur.

```

0  "" 7-5 : Character blend with background ""
10 SCREEN 1:COLOR 0,1:CLS
20 FOR I=70 TO 200 STEP 4
30   LINE(I,0)-(I+40,199),2
40 NEXT
50 LINE INPUT W$
60 FOR X=0 TO 103
70   FOR Y=0 TO 7
80     IF POINT(X,Y)=3
        THEN
            PSET(100+X,90+Y),3:PSET(110+X,120+Y),3

```

Hello there!

Hello there!

Hello there!

Fig. 7.4

```
ELSE
  PRESET(100+X,90+Y)
90 NEXT
95 NEXT
```

PROGRAMME 7.5

7.8 CARACTERES SANS LIMITATION DE COULEUR

Le mouvement réalisé dans les programmes 7.4 et 7.5 était fait à partir de caractères standards (couleur 3) dans une destination nouvelle non limitée, mais la couleur utilisée était toujours la couleur 3. Comme le traçage des caractères est fait en fait par l'instruction PSET, n'importe quelle couleur peut être utilisée, y compris la couleur du fond. Le programme 7.6 demande à la ligne 20 la couleur des caractères et copie deux fois la chaîne rentrée à la ligne 40 avec la couleur appropriée. La ligne 50 trace une boîte avec la couleur 3 pour rendre la couleur 0 visible

```
0 "" 7-6 : Character in any color ""
10 SCREEN 1:CLS
20 INPUT"Color ";COL
30 CLS
40 LINE INPUT W$
50 LINE(0,24)-(160,31),3,BF
```

```

60 FOR X=0 TO 159
70   FOR Y=0 TO 7
80     IF POINT(X,Y)=3
      THEN
        PSET(X,Y+24),COL:PSET(PSET(X,Y+32),COL
90   NEXT
95 NEXT

```

PROGRAMME 7.6

Avec cette méthode, il est même possible d'avoir des caractères de plus d'une couleur. Si on change la ligne 80 du programme 7.6 en :

```

80   IF POINT(X,Y)=3
      THEN
        PSET(X,Y+24),1+X MOD 3:
        PSET(X,Y+32),1+Y MOD 3

```

et qu'on efface la ligne 50, les caractères sont affichés en rayures horizontales et verticales de couleurs différentes.

Le programme 7.7 montre comment afficher des caractères noirs sur fond blanc en haute résolution. On voit une partie de l'écran sur la figure 7.5.

the red dying leaves of fall.

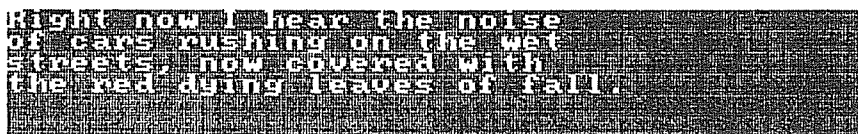


Fig. 7.5

```

0  °° 7-7 : Inverse characters °°
10 SCREEN 2:CLS
20 LINE(0,24)-(639,199),1,BF
30 FOR I=3 TO 25
40   LINE(0,0)-(639,7),0,BF
50   LOCATE 1,1:LINE INPUT W$
60   FOR X=0 TO 159
70     FOR Y=0 TO 7
80       IF POINT(X,Y)=1
        THEN
          PSET(X,Y+I*8),0
90     NEXT

```

93 NEXT

96 NEXT

PROGRAMME 7.7

7.9 CARACTERES AVEC OMBRE

La méthode que nous présentons dans ce paragraphe produit des lettres solides, c'est-à-dire des lettres avec un côté ombré qui les fait apparaître comme tri-dimensionnelles.

Si on copie un caractère avec une couleur différente de celle du fond et qu'on fasse une seconde copie un pixel à gauche et un pixel en haut avec une couleur différente de la première copie et du fond, le résultat sera un caractère qui apparaîtra en relief. La seconde lettre tracée couvrira la majeure partie de la première, mais la petite partie qui reste est celle qui produit l'effet d'ombre.

Le programme 7.8 balaie une chaîne de vingt caractères et la copie deux fois, de couleurs différentes, au centre de l'écran.

```
0  "" 7-8 : Characters with shadow ""
10 SCREEN 1:COLOR 0,0:CLS
20 LINE INPUT W$
30 FOR X=0 TO 159
40   FOR Y=0 TO 7
50     IF POINT(X,Y)=3
        THEN
            PSET(101+X,91+Y),2:PSET(100+X,90+Y),1
60   NEXT
70 NEXT
```

PROGRAMME 7.8

7.10 LES CARACTERES LARGES

Le détail remarquable de la haute résolution a malheureusement aussi l'inconvénient d'avoir des caractères qui sont difficiles à lire sur des écrans TV ou des moniteurs en basse résolution. En utilisant une version modifiée des programmes de coloration des caractères ou de déplacement des caractères des deux paragraphes précédents, nous allons produire en haute résolution des caractères qui sont normalement affichés en moyenne résolution.

Les modèles utilisés pour les caractères dans les deux modes graphiques sont les mêmes, la seule différence étant la plus petite largeur du pixel de haute résolution en haute résolution. Avec la fonction POINT, nous

balayons la zone où un caractère a été imprimé. Quand nous trouvons un pixel allumé nous traçons non pas un, mais deux points, côte à côte, au nouvel emplacement choisi. Les caractères résultants de ce processus sont exactement ceux de la moyenne résolution. La figure 7.6 montre à la fois des caractères étroits et des caractères larges.

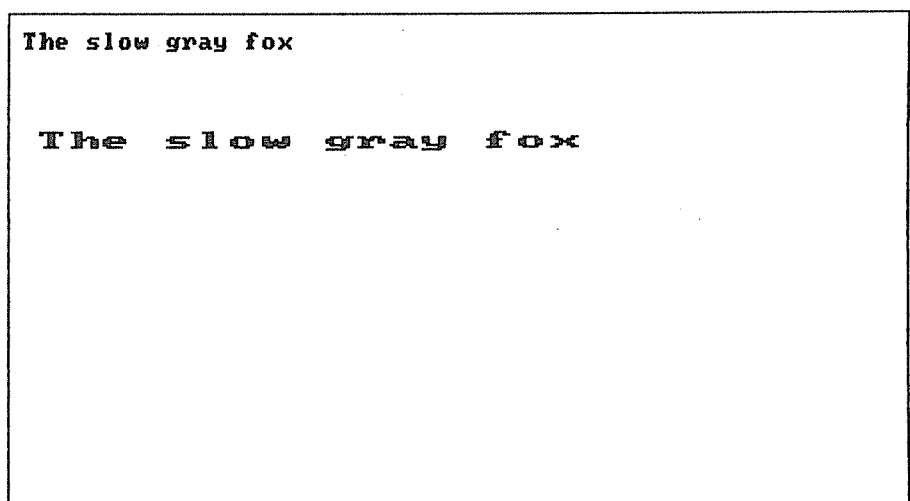


Fig. 7.6

Le programme 7.9 élargit horizontalement les caractères tapés à la ligne 20.

```
0  "" 7-9 : Wide characters ""
10 SCREEN 2:CLS
20 LINE INPUT W$
30 FOR X=0 TO 159
40   FOR Y=0 TO 7
50     IF POINT(X,Y)
        THEN
            PSET(2*X,Y+40):
            PSET(2*X+1,Y+40)
60   NEXT
70 NEXT
```

PROGRAMME 7.9

Bien qu'on puisse utiliser une technique similaire pour produire des petits caractères en moyenne résolution (de sorte que des lignes de 80 caractères soient possibles), le résultat n'est pas très net et le texte produit est d'habitude difficile à lire.

Notez qu'à la ligne 50 nous avons omis le IF POINT(X,Y)=1 (ou 3 en moyenne résolution) habituel. Dans cette nouvelle forme, la fonction POINT retourne VRAI quand la couleur est différente de 0 et FAUX sinon (voir appendice B).

7.11 LES GROSSES LETTRES

La méthode pour élargir les caractères vue au paragraphe précédent peut être étendue pour produire des caractères de n'importe quelles proportions, par exemple des caractères de largeur triple ou de hauteur double, de largeur double ou de hauteur triple, etc. Le programme 7.9 trace deux points chaque fois que la fonction POINT retourne VRAI. Comme maintenant nous ne savons pas à l'avance de combien chaque point va être élargi, nous utiliserons des blocs pleins au lieu d'une succession de PSETs. Ceci accélère également le programme. Nous permettons seulement l'agrandissement dans les directions horizontale et verticale, de sorte que chaque pixel soit reproduit comme un rectangle, donc notre choix d'un bloc (dessiné par LINE(X1,Y1)-(X2,Y2),COL,BF) marche correctement.

Le programme 7.10 demande en ligne 25 les facteurs d'agrandissement désirés dans les directions X et Y, et utilise un LINE INPUT pour mettre les caractères sur l'écran en partant du point en haut à gauche. la figure 7.7 montre les lettres produites avec HS=4 et VS=5.

```
0  "" 7-10 : Big letters ""
10 SCREEN 1:CLS
20 INPUT"Starting point (X,Y) ";STARTX,STARTY
25 INPUT"Scale (Horizontal,vertical) ";HS,VS
30 CLS:LINE INPUT W$
40 FOR X=0 TO 159
50   FOR Y=0 TO 7
60     COL=POINT(X,Y)
70     X1=STARTX+X*HS:Y1=STARTY+Y*VS
80     LINE(X1,Y1)-(X1+(HS-1),Y1+(VS-1)),COL,BF
90   NEXT
95 NEXT
```

PROGRAMME 7.10

7.12 CARACTERES N'AYANT PAS UNE ORIENTATION NORMALE

Il est possible de manipuler l'orientation des caractères en changeant tout simplement l'ordre dans lequel les points sont tracés (voir paragraphes 7.6 à 7.11). Comme la plupart des programmes qui manipulent

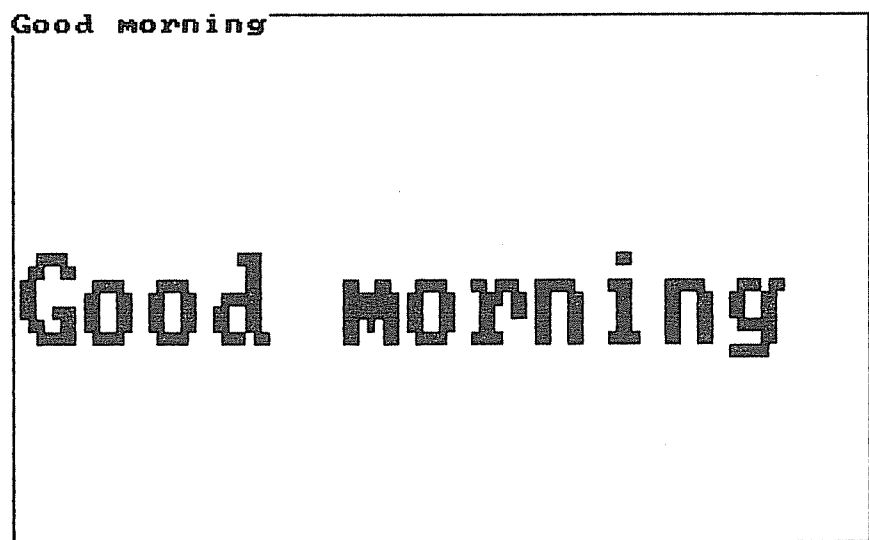


Fig. 7.7

l'orientation des caractères sont similaires, nous présentons le programme de base et ensuite la variation particulière pour chaque effet. Ces programmes mettent les caractères à des emplacements spécifiques, mais en changeant les constantes dans l'instruction PSET, on peut changer l'orientation de manière à réaliser des effets spéciaux n'importe où sur l'écran.

7.12.1 Miroir

Avec la méthode utilisée au programme 7.11, les caractères commencent sur le côté droit de l'écran et avancent vers la gauche. La figure 7.8 montre le résultat.

Agatha Christie Agatha Christie

Fig. 7.8

Just a sample.
Just a sample.

Fig. 7.9

```

0  °° 7-11 : Mirrored characters °°
10 SCREEN 1:COLOR 0,0:CLS
20 LINE INPUT W$
30 FOR X=0 TO 103
40   FOR Y=0 TO 7
50     IF POINT(X,Y)=3
        THEN
            PSET(319-X,8+Y)
60   NEXT
70 NEXT

```

PROGRAMME 7.11

7.12.2 La tête en bas

Ici les caractères sont renversés par rapport à l'axe des Y. Changez la ligne 50 du programme en :

```

50   IF POINT(X,Y)=3
        THEN
            PSET(X,20-Y)

```

La figure 7.9 montre les caractères la tête en bas.

7.12.3 Miroir renversé

Ici les caractères débutent sur le côté droit de l'écran et sont dessinés la tête en bas. Changez la ligne 50 du programme en :

```

50   IF POINT(X,Y)=3
        THEN
            PSET(319-X,20-Y)

```

La figure 7.10 montre cet effet.

Mirror on the wall

l l e m a h t u o j o r x i w

Fig. 7.10

7.12.4 Lecture verticale vers le haut

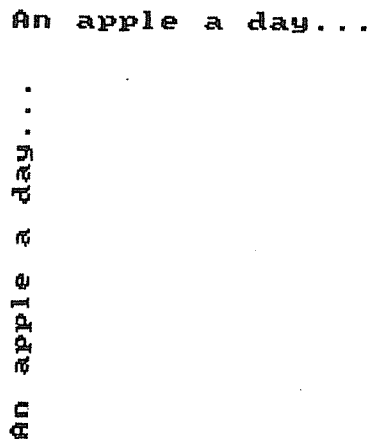
Ici les caractères se lisent verticalement en partant du bas vers le haut de l'écran. Changez la ligne 50 du programme 7.11 en :

```

50    IF POINT(X,Y)=3
      THEN
        PSET(Y,160-X)

```

La figure 7.11 montre les caractères se lisant verticalement vers le haut.



An apple a day...

Fig. 7.11

7.12.5 Lecture verticale vers le bas

Ici les caractères se lisent verticalement en partant du haut vers le bas de l'écran. Changez la ligne 50 du programme 7.11 en :

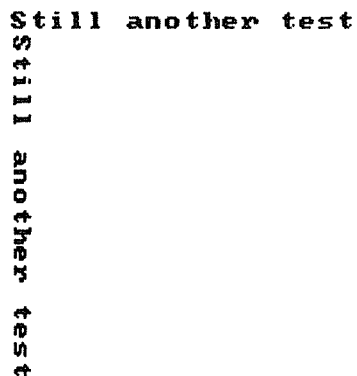
```

50    IF POINT(X,Y)=3
      THEN
        PSET(7-Y,10+X)

```

La figure 7.12 montre quelques caractères se lisant verticalement vers le bas.

On peut utiliser cette méthode pour faire des copies sur papier de documents très grands. Le nombre de caractères que n'importe quelle imprimante peut imprimer sur une ligne est limité. Toutefois si le document est mis verticalement et que les écrans soient imprimés successivement, il n'y a pas de limite pour la longueur des lignes.



Still another test

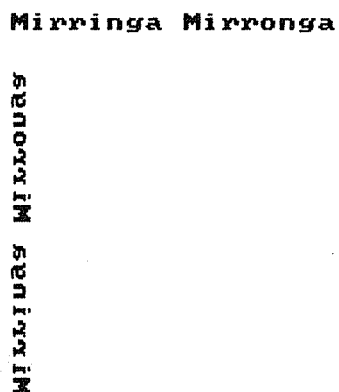
Fig. 7.12

7.12.6 Lecture verticale vers le haut inversée

Ici les caractères se lisent verticalement, du bas vers le haut de l'écran, les caractères étant inversés. Changez la ligne 50 du programme 7.11 en :

```
50  IF POINT(X,Y)=3
      THEN
      PSET(7-Y,160-X)
```

La figure 7.13 montre des caractères inversés se lisant vers le haut.



Mirringa Mirronga

Fig. 7.13

7.12.7 Lecture verticale vers le bas inversée

Ici les caractères se lisent verticalement du haut vers le bas de l'écran, les caractères étant inversés. Changez la ligne 50 du programme 7.11 en :

```

50    IF POINT(X,Y)=3
      THEN
        PSET(Y,10+X)

```

La figure 7.14 montre des caractères inversés se lisant vers le bas.

ancho panza

Fig. 7.14

7.12.8 Caractères inversés individuellement

Ici les caractères sont placés horizontalement de gauche à droite mais chaque caractère est renversé par rapport à l'axe des Y. Le programme 7.12 accomplit cet effet particulier en balayant caractère par caractère, et non la totalité de la chaîne en une seule fois comme dans les programmes précédents de ce paragraphe.

```

0  ' 7-12 : Individually inverted char. °
10 SCREEN 1:COLOR 0,0:CLS
20 LINE INPUT W$
30 FOR CHAR=0 TO 20
40   FOR X=0 TO 7
50     FOR Y=0 TO 7
60       IF POINT(CHAR*8+X,Y)=3
          THEN
            PSET(CHAR*8+7-X,10+Y)
70     NEXT
80   NEXT
90 NEXT

```

PROGRAMME 7.12

La figure 7.15 montre une chaîne dont les caractères ont été inversés individuellement.

Fusagasuga
72565206

Fig. 7.15

7.13 CURSEUR LOGICIEL

Le curseur de texte est particulièrement utile pour montrer la position du champ d'entrée, et pour faire savoir à l'utilisateur qu'une entrée est attendue. En moyenne et haute résolutions, l'instruction INPUT allume un curseur qui ne clignote pas⁴, mais qui est acceptable. Afin d'avoir un contrôle absolu sur l'écran⁵, on utilise l'instruction LINE INPUT ou les fonctions INPUT\$(1) et INKEY\$. Malheureusement ces fonctions ne produisent pas du tout de curseur, et l'utilisateur peut être désorienté du fait qu'il ne sait pas où se trouve le champ d'entrée, ou même si une entrée est attendue ou non. Dans ce paragraphe nous expliquerons comment produire un curseur clignotant logiciel.

La fonction INKEY\$ lit le clavier sans interrompre le déroulement du programme. Quand on la rencontre dans un programme, si une touche a été pressée et pas encore traitée, le caractère correspondant est retourné et le programme continue. Si il n'y a pas de caractères attendant au clavier, le programme continue de toutes façons, et INKEY\$ retourne la chaîne nulle (« »). Ceci diffère d'avec les autres fonctions ou instructions d'entrée dans lesquelles le programme s'arrête complètement jusqu'à ce que la rentrée soit terminée. On peut utiliser le sous-programme 1000 (programme 7.13) pour rechercher un caractère à partir du clavier d'une manière similaire à la fonction INPUT\$(1) avec la différence qu'elle produit un curseur clignotant. La ligne 1040 allume le curseur en dessinant un bloc de la couleur 3, exactement de 8 pixels sur 8 (la taille d'une cellule de caractère), et l'étend en dessinant un bloc de la même taille avec la couleur du fond. En changeant la valeur avec laquelle est comparé le compteur C à la ligne 1030 (20 dans notre cas), le curseur clignotera plus ou moins vite.

```
1000 'Input function
1005 X1=(X-1)*8:Y1=(Y-1)*8
1010 W$=INKEY$
1020 IF W$<>" "
      THEN
        RETURN
1030 C=C+1:
      IF C<20
        THEN
          1010
1040 C=0:FL=NOT FL:
```

```

IF FL
  THEN
    LINE(X1,Y1)-(X1+7,Y1+7),3,BF:GOTO 1010
  ELSE
    LINE(X1,Y1)-(X1+7,Y1+7),0,BF:GOTO 1010

```

PROGRAMME 7.13

7.14 SOUS-PROGRAMME D'ENTREE AVEC UN CURSEUR CLIGNOTANT

Nous présentons ici un sous-programme d'entrée simple mais complet. Il comprend l'effacement en arrière d'un caractère, l'utilisation de RETURN pour signaler la fin des rentrées, un curseur clignotant et le contrôle de la longueur du champ d'entrée⁶. Le programme 7.14 doit être fusionné avec le programme 7.13, le sous-programme du curseur clignotant.

```

0  "" 7-14 : Controlled input ""
10 SCREEN 1:CLS
20 X=10:Y=15:INLEN=15
30 GOSUB 500:NAME.$=BUFF$
40 END
500 ' Input subroutine
510 BUFF$="":LOCATE Y,X
520 ' Fetch a character
530 GOSUB 1000
540 ' Is character <return> ?
550 IF W$=CHR$(13)
    THEN
      RETURN
560 ' Is character a <backspace> ?
570 IF W$=CHR$(8)
    THEN
      IF BUFF$=""
        THEN
          530
        ELSE
          PRINT " ";CHR$(29);CHR$(29);:X=X-1:
          BUFF$=LEFT$(BUFF$,LEN(BUFF$)-1):GOTO 5030
580 ' Is character printable ?
590 IF W$<" " OR W$>CHR$(127)
    THEN
      5030
600 ' Is there still room for a new character ?

```



```

610 IF LEN(BUFF$)=INLEN
    THEN
        5030
    ELSE
        PRINT W$;:BUFF$=BUFF$+W$:X=X+1:GOTO 5030

```

PROGRAMME 7.14

Le sous-programme 500 a besoin des coordonnées du caractère du champ d'entrée (X,Y), et la longueur désirée (INLEN). Si par exemple X = 5, Y = 3 et INLEN = 10, le curseur commencera à clignoter à la position (5,3) et le sous-programme d'entrée acceptera jusqu'à 10 caractères.

On utilise BUFF\$ pour mémoriser l'état en cours du champ d'entrée, c'est-à-dire les caractères qui ont été tapés et acceptés. A la ligne 510, BUFF\$ est vidé pour éviter des conflits avec des champs d'entrée précédents. L'instruction LOCATE Y,X place le curseur au début du champ d'entrée.

La ligne 530 va chercher un caractère en utilisant le sous-programme 1000, ou le programme 7-13, qui crée le curseur clignotant. Si le caractère est un retour de chariot (CHR\$(13)), la ligne 550 termine le sous-programme. Notez que le <RETURN> (CHR\$(13)) n'est pas compris dans BUFF\$. Si le caractère est un espace en arrière (CHR\$(8)), il y a deux cas à traiter :

Quand BUFF\$ est vide, il ne faut rien faire.

Quand BUFF\$ n'est pas vide, un caractère doit être effacé de BUFF\$ et de l'écran, et X doit être décrémenté.

L'instruction BUFF\$ = LEFT\$(BUFF\$,LEN(BUFF\$)-1) prend en charge le caractère le plus à droite de BUFF\$. Pour mettre à jour l'écran, le programme imprime un blanc, effaçant donc le curseur clignotant (s'il y en avait un). Imprimer CHR\$(29) a l'effet de déplacer le curseur sur la gauche, et on a besoin de deux de ces caractères pour laisser le curseur à la position du dernier caractère tapé, lequel ne doit pas être effacé. Le curseur clignotant le prendra en charge.

Si la ligne 590 découvre que le caractère n'est pas imprimable⁷, elle le rejette et revient à la ligne 530 pour aller en chercher un nouveau. Si non, la ligne 710 vérifie s'il y a ou non toujours de la place dans BUFF\$. Si par exemple la longueur spécifiée est 20 et que BUFF\$ a déjà 20 caractères, un nouveau caractère devra être rejeté. S'il y a de la place, le caractère est imprimé et rajouté à BUFF\$, et X est incrémenté pour mettre à jour la position du curseur clignotant.

7.15 IMPRESSION SUR LA DERNIERE POSITION

Quand un caractère est imprimé sur la position la plus à droite (40 en moyenne résolution, 80 en haute résolution) des lignes 24 et 25, l'écran est remonté : bien qu'il soit nécessaire d'afficher le texte séquentiellement lorsque vous utilisez l'instruction PRINT, c'est gênant lorsque vous avez besoin de ces positions. Supposez que nous voulions un programme pour entourer l'écran du cadre que l'on voit à la figure 7.16.

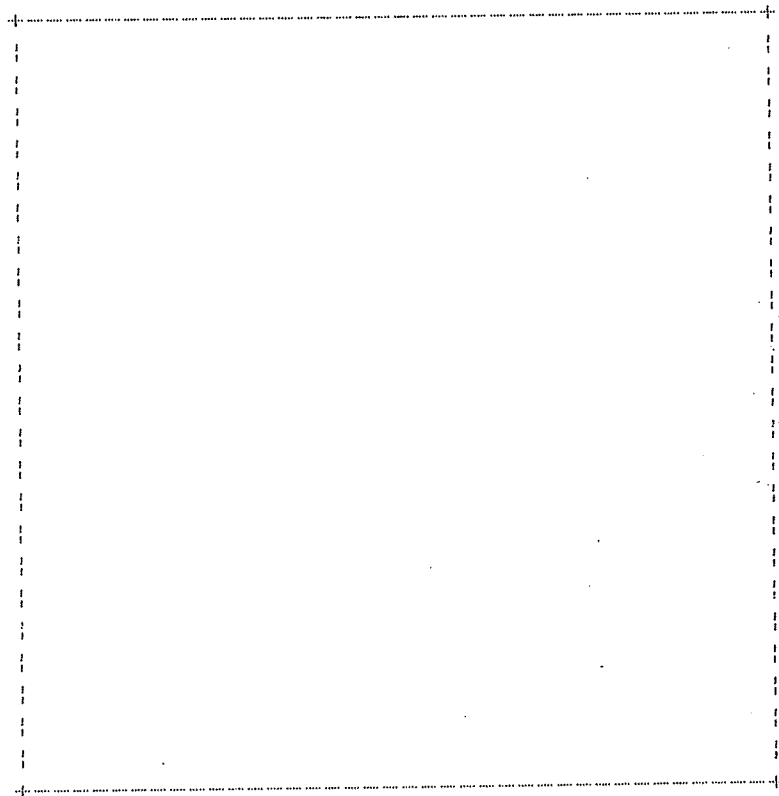


Fig. 7.16

Nous voulons que le dernier signe plus se trouve à la position (25,40), et le « | » de la ligne précédente à la position (24,40). Comme il n'est pas possible d'imprimer des caractères sur ces positions sans remonter l'écran, il semblera impossible de dessiner ce cadre. Traités comme des caractères, le « + » et le « | » sont un problème, mais s'ils sont considérés comme une suite spéciale de points graphiques, ils peuvent être placés n'importe où sur l'écran. Le programme 7.15 imprime le signe plus à la position (1,1) et le copie en utilisant la fonction POINT, point par point, jusqu'à sa destination finale. Pour calculer l'emplacement de la destination

nous pouvons utiliser la formule 7.1. Comme la position du « + » est (40,25) le coin en haut à gauche de la cellule du caractère est en coordonnées graphiques :

$$X = (40-1) \cdot 8 = 312$$

$$Y = (25-1) \cdot 8 = 192$$

Le même processus est appliqué au « | » en changeant seulement la destination, position (40,24). Le reste de l'écran peut être imprimé normalement.

```

0  "" 7-15 : Print in last position ""
10 SCREEN 1:COLOR 0,0:CLS
20 W$="+":X=40:Y=25:GOSUB 500
30 W$="|":X=40:Y=24:GOSUB 500
40 LOCATE 1,1:PRINT"+";STRING$(38,"-");"+"
50 FOR I=2 TO 24
60   LOCATE I,1:PRINT"|";
70   IF I<24
      THEN
        LOCATE I,40:PRINT"|";
80 NEXT
90 LOCATE 25,1:PRINT "+";STRING$(38,"-");
100 GOTO 100
500 ' Copy character into position.
510 LOCATE 1,1:PRINT W$;
520 ' X,Y are character coord. of destination
530 X1=(X-1)*8:Y1=(Y-1)*8
540 FOR I=0 TO 7
550   FOR J=0 TO 7
560     IF POINT(I,J)=3
        THEN
          PSET(X1+I,Y1+J)
        ELSE
          PRESET(X1+I,Y1+J)
570   NEXT
580 NEXT
590 RETURN

```

PROGRAMME 7.15

On peut utiliser la même technique en haute résolution. Au chapitre 8 nous étudierons une méthode plus efficace pour réaliser la même chose.

7.16 EFFACEMENT JUSQU'EN FIN DE LIGNE

Très fréquemment il est nécessaire d'effacer tous les caractères à partir de la position en cours du curseur jusqu'au bord droit de l'écran. Par exemple lorsque vous faites des entrées et que vous voulez rejeter des réponses erronées en plaçant le curseur juste après l'entrée initiale, il est nécessaire d'effacer les caractères précédents pour laisser le champ libre pour une nouvelle entrée. Le programme 7.16 n'efface pas le champ et montre comment la situation peut être désorientante.

```
0  "" 7-16 : Confusing input ""
10 CLS
20 LOCATE 10,15
30 PRINT"Agree ";INPUT W$
40 IF W$<>"More or less"
    THEN
        20
```

PROGRAMME 7.16

Pour les 23 premières lignes, l'effacement jusqu'à la fin de la ligne peut être facilement accompli par cette instruction :

```
PRINT SPACE$(40-POS(0)+1)
```

Le programme 7.17 utilise cette instruction pour corriger le problème du programme 7.16.

```
0  "" 7-17 : Confusing input corrected ""
10 CLS
20 LOCATE 10,15
30 PRINT"Agree ";GOSUB 1000:INPUT W$
40 IF W$<>"More or less"
    THEN
        20
50 END
1000 ' Clear to end of line
1010 XTEMP=POS(0):YTEMP=CSRLIN
1020 PRINT SPACE$(40-XTEMP+1);
1030 LOCATE YTEMP,XTEMP
1040 RETURN
```

PROGRAMME 7.17

On ne peut pas utiliser cette méthode avec les lignes 24 et 25 parce que l'écran est remonté quand le caractère le plus à droite est imprimé.

Nous aurons recours aux instructions graphiques parce qu'elles peuvent être utilisées n'importe où sur l'écran, même dans ces emplacements interdits aux textes.

Nous rappellerons qu'avec la formule 7.1 nous pouvons calculer le point en haut à gauche de la cellule de caractère à la position (X,Y). Sachant que de telles cellules ont exactement 8 pixels de large, nous pouvons dessiner un bloc plein (avec l'instruction LINE) à partir de ce point jusqu'au bord droit de l'écran, un bloc large exactement de huit pixels qui effacera tous les caractères jusqu'à la fin de la ligne, même aux lignes 24 et 25. Notez que dans le programme 7.18 le curseur de texte ne doit pas être touché, donc le sous-programme 1000 n'a pas à le repositionner. INPUT et LINE INPUT produisent un retour chariot automatique. Si on ajoute un point virgule entre LINE INPUT et la variable, ce retour ne sera pas répété à la ligne 24. Malheureusement il est toujours répété à la ligne 25 et la seule manière d'utiliser efficacement le sous-programme 1000 dans cette ligne est de contrôler ce qui est entré avec une méthode semblable à celle utilisée au programme 7.14.

```
0  ** 7-18 : Erase to end of line **
10 SCREEN 1:CLS
20 LOCATE 10,15:PRINT"Agree ";:GOSUB 1000:LINE INPUT;W$
30 IF W$<>"More or less"
    THEN
        20
40 END
1000 ' Erase to end of line with block
1010 XTEMP=POS(0):YTEMP=CSRLIN
1020 X=(XTEMP-1)*8:Y=(YTEMP-1)*8
1030 LINE(X,Y)-(319,Y+7),0,BF
1040 RETURN
```

PROGRAMME 7.18

7.17 EFFACEMENT JUSQU'A LA FIN DE L'ECRAN

Si vous avez essayé le programme 7.17 ou 7.18 avec de longues réponses à la question « Agree ? », vous devez avoir remarqué que lorsque certains caractères sont imprimés sur la ligne suivante, effacer jusqu'à la fin de la ligne n'est pas suffisant pour garantir une entrée correcte. Une des nombreuses solutions est d'effacer jusqu'à la fin de l'écran, c'est-à-dire effacer à partir du curseur jusqu'à la fin de la ligne et effacer également toutes les lignes situées en dessous. Nous utiliserons de nouveau la boîte du programme 7.18, mais cette fois nous en ajouterons une seconde assez grande pour effacer le reste de l'écran. La ligne 15 du programme 7.19 remplit l'écran avec des « et » commerciaux (&) (en prenant soin de ne pas

toucher les emplacements (40,24) et (40,25) pour rendre l'effet « effacement jusqu'à la fin de l'écran » perceptible. Voyez que vous pouvez taper autant de caractères que vous voulez (pour autant que vous n'atteignez pas l'emplacement (40,24) et le sous-programme 1000 les effacera.

```

0  °° 7-19 : Erase to end of screen °°
10 SCREEN 1:CLS
15 FOR I=1 TO 25:
    LOCATE I,1:PRINT STRING$(39,"&");:
NEXT
20 LOCATE 15,15:PRINT"Agree ";:GOSUB 1000:
    LINE INPUT;W$
30 IF W$<>"More or less"
    THEN
        20
40 END
1000 ' Clear to end of screen
1010 XTEMP=POS(0):YTEMP=CSRLIN
1020 X=(XTEMP-1)*8:Y=(YTEMP-1)*8
1030 LINE(X,Y)-(319,Y+7),0,BF
1040 LINE(0,Y+8)-(319,199),0,BF
1050 RETURN

```

PROGRAMME 7.19

7.18 L'ALPHABET

Nous présentons ici un programme pour créer un alphabet en grandes lettres, dans lequel chaque caractère est dessiné dans une cellule de 51X61 pixels. Bien que les lettres puissent être dimensionnées avec la méthode décrite au paragraphe 7.11, nous souhaitons que vous utilisiez le programme 7.20 comme exemple pour créer vos propres ensembles et dimensions de caractères. On peut voir les lettres obtenues à la figure 7.17.

```

0  °° 7-20 : Graphic alphabet °°
10 SCREEN 1:CLS
20 X=0:Y=0
30 W$=INPUT$(1):
    IF W$=" "
        THEN
            50
40 IF W$<"a"AND W$>"z"
    THEN
        10
    ELSE
        ON ASC(W$)-64 GOSUB 100,200,300,400,500,
            600,700,800,900,1000,1100,1200,1300,
            1400,1500,1600,1700,1800,1900,2000,

```

```

2100,2200,2300,2400,2500,2600
50 X=X+65:
  IF X>290
  THEN
    X=0:Y=Y+68:
    IF Y>200
    THEN
      END
60 GOTO 30
100 *** A **
115 LINE (0+X,60+Y)-(20+X,0+Y)
120 LINE (30+X,0+Y)-(50+X,60+Y)
125 LINE (20+X,0+Y)-(30+X,0+Y)
130 LINE (0+X,60+Y)-(10+X,60+Y)
135 LINE (40+X,60+Y)-(50+X,60+Y)
140 LINE (18+X,35+Y)-(32+X,35+Y)
145 LINE (16+X,45+Y)-(35+X,45+Y)
150 LINE (10+X,60+Y)-(15+X,45+Y),3
155 LINE (35+X,45+Y)-(40+X,60+Y),3
160 LINE (18+X,35+Y)-(25+X,15+Y),3
165 LINE (25+X,15+Y)-(31+X,35+Y),3
170 PAINT (25+X,2+Y),2,3
175 RETURN
200 *** B **
205 LINE (0+X,0+Y)-(0+X,60+Y),3
210 LINE (0+X,0+Y)-(30+X,0+Y),3
215 LINE (0+X,60+Y)-(32+X,60+Y),3
220 CIRCLE (31+X,14+Y),14,3,4.71,1.57,1
225 CIRCLE (22+X,16+Y),7,3,4.71,1.57,1
230 LINE (12+X,9+Y)-(22+X,9+Y)
235 LINE (12+X,23+Y)-(22+X,23+Y)
240 LINE (12+X,9+Y)-(12+X,23+Y)
245 CIRCLE (33+X,44+Y),16,3,4.71,1.57,1
250 CIRCLE (26+X,43+Y),8,3,4.71,1.57,1
255 LINE (12+X,35+Y)-(25+X,35+Y)
260 LINE (12+X,51+Y)-(25+X,51+Y)
265 LINE (12+X,35+Y)-(12+X,51+Y)
270 PAINT (1+X,1+Y),2,3
275 RETURN
300 *** C **
305 CIRCLE (25+X,25+Y),25,3,0,3.1416,1
310 CIRCLE (25+X,25+Y),15,3,0,3.1416,1
315 CIRCLE (25+X,35+Y),25,3,3.1416,6.28,1
320 CIRCLE (25+X,35+Y),15,3,3.1416,6.28,1
325 LINE (0+X,25+Y)-(0+X,35+Y)
330 LINE (9+X,25+Y)-(9+X,35+Y)
335 LINE (40+X,25+Y)-(50+X,25+Y)
340 LINE (40+X,35+Y)-(50+X,35+Y)
345 PAINT (2+X,30+Y),2,3
350 RETURN
400 *** D **
405 LINE (0+X,0+Y)-(0+X,60+Y)
410 LINE (0+X,0+Y)-(30+X,0+Y)
415 LINE (0+X,60+Y)-(30+X,60+Y)
420 CIRCLE (28+X,20+Y),20,3,0,1.57,1
425 CIRCLE (28+X,40+Y),20,3,4.71,0,1
430 LINE (49+X,20+Y)-(49+X,40+Y)
435 CIRCLE (23+X,23+Y),12,3,0,1.57,1
440 CIRCLE (23+X,37+Y),12,3,4.71,0,1
445 LINE (36+X,23+Y)-(36+X,37+Y)
450 LINE (12+X,11+Y)-(23+X,11+Y)

```

```

455 LINE(12+X,49+Y)-(23+X,49+Y)
460 LINE(12+X,11+Y)-(12+X,49+Y)
465 PAINT(2+X,2+Y),2,3
470 RETURN
500 ' ** E **
505 LINE(0+X,0+Y)-(50+X,0+Y)
510 LINE(0+X,0+Y)-(0+X,60+Y)
515 LINE(0+X,60+Y)-(50+X,60+Y)
520 LINE(12+X,12+Y)-(50+X,12+Y)
525 LINE(50+X,0+Y)-(50+X,12+Y)
530 LINE(12+X,49+Y)-(50+X,49+Y)
535 LINE(50+X,49+Y)-(50+X,60+Y)
540 LINE(12+X,26+Y)-(35+X,26+Y)
545 LINE(12+X,37+Y)-(35+X,37+Y)
550 LINE(35+X,26+Y)-(35+X,37+Y)
555 LINE(12+X,12+Y)-(12+X,26+Y)
560 LINE(12+X,37+Y)-(12+X,49+Y)
565 PAINT(2+X,2+Y),2,3
570 RETURN
600 ' ** F **
605 LINE(0+X,0+Y)-(50+X,0+Y)
610 LINE(0+X,0+Y)-(0+X,60+Y)
615 LINE(0+X,60+Y)-(12+X,60+Y)
620 LINE(12+X,12+Y)-(50+X,12+Y)
625 LINE(50+X,0+Y)-(50+X,12+Y)
630 LINE(12+X,26+Y)-(35+X,26+Y)
635 LINE(12+X,37+Y)-(35+X,37+Y)
640 LINE(35+X,26+Y)-(35+X,37+Y)
645 LINE(12+X,12+Y)-(12+X,26+Y)
650 LINE(12+X,37+Y)-(12+X,60+Y)
655 PAINT(2+X,2+Y),2,3
660 RETURN
700 ' ** G **
705 CIRCLE(25+X,25+Y),25,3,0,3.1416,1
710 CIRCLE(25+X,25+Y),15,3,0,3.1416,1
715 CIRCLE(25+X,35+Y),25,3,3.1416,5.45,1
720 CIRCLE(25+X,35+Y),15,3,3.1416,5.7,1
725 LINE(0+X,25+Y)-(0+X,35+Y)
730 LINE(9+X,25+Y)-(9+X,35+Y)
735 LINE(40+X,25+Y)-(50+X,25+Y)
740 LINE(30+X,35+Y)-(50+X,35+Y)
745 LINE(30+X,43+Y)-(37+X,43+Y)
750 LINE(30+X,35+Y)-(30+X,43+Y)
755 LINE(50+X,35+Y)-(50+X,60+Y)
760 LINE(42+X,54+Y)-(42+X,60+Y)
765 LINE(42+X,60+Y)-(50+X,60+Y)
770 PAINT(2+X,30+Y),2,3
775 RETURN
800 ' ** H **
805 LINE(0+X,0+Y)-(0+X,60+Y)
810 LINE(50+X,0+Y)-(50+X,60+Y)
815 LINE(0+X,0+Y)-(12+X,0+Y)
820 LINE(0+X,60+Y)-(12+X,60+Y)
825 LINE(38+X,0+Y)-(50+X,0+Y)
830 LINE(38+X,60+Y)-(50+X,60+Y)
835 LINE(12+X,26+Y)-(38+X,26+Y)
840 LINE(12+X,37+Y)-(38+X,37+Y)
845 LINE(12+X,0+Y)-(12+X,26+Y)
850 LINE(12+X,37+Y)-(12+X,60+Y)
855 LINE(38+X,0+Y)-(38+X,26+Y)

```



```

860 LINE (38+X,37+Y)-(38+X,60+Y)
865 PAINT (2+X,2+Y),2,3
870 RETURN
900 *** I **
905 LINE (18+X,0+Y)-(35+X,0+Y)
910 LINE (18+X,60+Y)-(35+X,60+Y)
915 LINE (18+X,0+Y)-(18+X,8+Y)
920 LINE (35+X,0+Y)-(35+X,8+Y)
925 LINE (18+X,52+Y)-(18+X,60+Y)
930 LINE (35+X,52+Y)-(35+X,60+Y)
935 LINE (22+X,8+Y)-(22+X,52+Y)
940 LINE (31+X,8+Y)-(31+X,52+Y)
945 LINE (18+X,8+Y)-(22+X,8+Y)
950 LINE (31+X,8+Y)-(35+X,8+Y)
955 LINE (18+X,52+Y)-(22+X,52+Y)
960 LINE (31+X,52+Y)-(35+X,52+Y)
965 PAINT (25+X,2+Y),2,3
970 RETURN
1000 *** J **
1005 CIRCLE (25+X,44+Y),16,3,3.14,6.28,1
1010 CIRCLE (25+X,44+Y),7,3,3.14,6.28,1
1015 LINE (41+X,0+Y)-(41+X,43+Y)
1020 LINE (32+X,8+Y)-(32+X,43+Y)
1025 LINE (9+X,43+Y)-(18+X,43+Y)
1030 LINE (27+X,0+Y)-(27+X,8+Y)
1035 LINE (27+X,8+Y)-(31+X,8+Y)
1040 LINE (27+X,0+Y)-(41+X,0+Y)
1045 PAINT (31+X,2+Y),2,3
1050 RETURN
1100 *** K **
1105 LINE (0+X,0+Y)-(0+X,60+Y)
1110 LINE (0+X,0+Y)-(12+X,0+Y)
1115 LINE (0+X,60+Y)-(12+X,60+Y)
1120 LINE (38+X,60+Y)-(12+X,29+Y)
1125 LINE (12+X,26+Y)-(38+X,0+Y)
1130 LINE (24+X,26+Y)-(50+X,0+Y)
1135 LINE (50+X,60+Y)-(23+X,27+Y)
1140 LINE (12+X,0+Y)-(12+X,25+Y)
1145 LINE (12+X,29+Y)-(12+X,60+Y)
1150 LINE (38+X,60+Y)-(50+X,60+Y)
1155 LINE (38+X,0+Y)-(50+X,0+Y)
1160 PAINT (2+X,2+Y),2,3
1165 RETURN
1200 *** L **
1205 LINE (0+X,0+Y)-(0+X,60+Y)
1210 LINE (0+X,0+Y)-(12+X,0+Y)
1215 LINE (0+X,60+Y)-(50+X,60+Y)
1220 LINE (12+X,0+Y)-(12+X,48+Y)
1225 LINE (12+X,48+Y)-(50+X,48+Y)
1230 LINE (50+X,48+Y)-(50+X,60+Y)
1235 PAINT (2+X,2+Y),2,3
1240 RETURN
1300 *** M **
1305 LINE (0+X,0+Y)-(0+X,60+Y)
1310 LINE (50+X,0+Y)-(50+X,60+Y)
1315 LINE (0+X,0+Y)-(10+X,0+Y)
1320 LINE (0+X,60+Y)-(10+X,60+Y)
1325 LINE (40+X,0+Y)-(50+X,0+Y)
1330 LINE (40+X,60+Y)-(50+X,60+Y)
1335 LINE (10+X,15+Y)-(10+X,60+Y)

```

```

1340 LINE (40+X,15+Y)-(40+X,60+Y)
1345 LINE (10+X,15+Y)-(25+X,37+Y)
1350 LINE (25+X,37+Y)-(40+X,15+Y)
1355 LINE (10+X,0+Y)-(25+X,22+Y)
1360 LINE (25+X,22+Y)-(40+X,0+Y)
1365 PAINT (2+X,2+Y),2,3
1370 RETURN
1400 '** N **
1405 LINE (0+X,0+Y)-(0+X,60+Y)
1410 LINE (50+X,0+Y)-(50+X,60+Y)
1415 LINE (0+X,0+Y)-(10+X,0+Y)
1420 LINE (0+X,60+Y)-(10+X,60+Y)
1425 LINE (40+X,0+Y)-(50+X,0+Y)
1430 LINE (40+X,60+Y)-(50+X,60+Y)
1435 LINE (10+X,15+Y)-(10+X,60+Y)
1440 LINE (40+X,0+Y)-(40+X,44+Y)
1445 LINE (10+X,0+Y)-(40+X,44+Y)
1450 LINE (10+X,15+Y)-(40+X,60+Y)
1455 PAINT (2+X,2+Y),2,3
1460 RETURN
1500 '** O **
1505 CIRCLE (25+X,25+Y),25,3,0,3.1416,1
1510 CIRCLE (25+X,25+Y),15,3,0,3.1416,1
1515 CIRCLE (25+X,35+Y),25,3,3.1416,6.28,1
1520 CIRCLE (25+X,35+Y),15,3,3.1416,6.28,1
1525 LINE (0+X,25+Y)-(0+X,35+Y)
1530 LINE (9+X,25+Y)-(9+X,35+Y)
1535 LINE (50+X,25+Y)-(50+X,35+Y)
1540 LINE (41+X,25+Y)-(41+X,35+Y)
1545 PAINT (2+X,30+Y),2,3
1550 RETURN
1600 '** P **
1605 CIRCLE (30+X,20+Y),20,3,4.71,1.57,1
1610 LINE (0+X,0+Y)-(0+X,60+Y)
1615 LINE (0+X,0+Y)-(30+X,0+Y)
1620 LINE (0+X,60+Y)-(12+X,60+Y)
1625 LINE (12+X,40+Y)-(33+X,40+Y)
1630 LINE (12+X,40+Y)-(12+X,60+Y)
1635 CIRCLE (25+X,20+Y),10,3,4.71,1.57,1
1640 LINE (12+X,10+Y)-(25+X,10+Y)
1645 LINE (12+X,30+Y)-(25+X,30+Y)
1650 LINE (12+X,10+Y)-(12+X,30+Y)
1655 PAINT (2+X,2+Y),2,3
1660 RETURN
1700 '** Q **
1705 CIRCLE (25+X,25+Y),25,3,0,3.1416,1
1710 CIRCLE (25+X,25+Y),15,3,0,3.1416,1
1715 CIRCLE (25+X,35+Y),25,3,3.1416,5.35,1
1720 CIRCLE (25+X,35+Y),25,3,5.74,6.28,1
1725 CIRCLE (25+X,35+Y),15,3,3.1416,5.23,1
1730 CIRCLE (25+X,35+Y),15,3,5.97,6.28,1
1735 LINE (0+X,25+Y)-(0+X,35+Y)
1740 LINE (9+X,25+Y)-(9+X,35+Y)
1745 LINE (50+X,25+Y)-(50+X,35+Y)
1750 LINE (41+X,25+Y)-(41+X,35+Y)
1755 LINE (35+X,37+Y)-(38+X,40+Y)
1760 LINE (28+X,43+Y)-(32+X,47+Y)
1765 LINE (34+X,36+Y)-(28+X,42+Y)
1770 LINE (40+X,55+Y)-(43+X,58+Y)
1775 LINE (47+X,48+Y)-(50+X,51+Y)
1780 LINE (50+X,51+Y)-(44+X,57+Y)

```

```

1785 PAINT(25+X,2+Y),2,3
1790 RETURN
1800 *** R **
1805 CIRCLE(30+X,20+Y),20,3,4.71,1.57,1
1810 LINE(0+X,0+Y)-(0+X,60+Y)
1815 LINE(0+X,0+Y)-(30+X,0+Y)
1820 LINE(0+X,60+Y)-(12+X,60+Y)
1825 LINE(12+X,40+Y)-(17+X,40+Y)
1830 LINE(12+X,40+Y)-(12+X,60+Y)
1835 CIRCLE(25+X,20+Y),10,3,4.71,1.57,1
1840 LINE(12+X,10+Y)-(25+X,10+Y)
1845 LINE(12+X,30+Y)-(25+X,30+Y)
1850 LINE(12+X,10+Y)-(12+X,30+Y)
1855 LINE(18+X,40+Y)-(30+X,60+Y)
1860 LINE(30+X,40+Y)-(42+X,60+Y)
1865 LINE(30+X,60+Y)-(41+X,60+Y)
1870 PAINT(2+X,2+Y),2,3
1875 RETURN
1900 *** S **
1905 CIRCLE(25+X,18+Y),18,3,0,4.71,1
1910 CIRCLE(25+X,18+Y),8,3,0,4.71,1
1915 CIRCLE(25+X,43+Y),17,3,3.1416,1.57,1
1920 CIRCLE(25+X,42+Y),6,3,3.1416,1.57,1
1925 LINE(33+X,19+Y)-(43+X,19+Y)
1930 LINE(8+X,42+Y)-(18+X,42+Y)
1935 PAINT(25+X,2+Y),2,3
1940 RETURN
2000 *** T **
2005 LINE(0+X,0+Y)-(50+X,0+Y)
2010 LINE-(50+X,12+Y)
2015 LINE-(30+X,12+Y)
2020 LINE-(30+X,60+Y)
2025 LINE-(19+X,60+Y)
2030 LINE-(19+X,12+Y)
2035 LINE-(0+X,12+Y)
2040 LINE-(0+X,0+Y)
2045 PAINT(2+X,2+Y),2,3
2050 RETURN
2100 *** U **
2105 CIRCLE(25+X,35+Y),25,3,3.14,6.28,1
2110 CIRCLE(25+X,35+Y),15,3,3.14,6.28,1
2115 LINE(0+X,35+Y)-(0+X,0+Y)
2120 LINE-(10+X,0+Y)
2125 LINE-(10+X,35+Y)
2130 LINE(50+X,35+Y)-(50+X,0+Y)
2135 LINE-(40+X,0+Y)
2140 LINE-(40+X,35+Y)
2145 PAINT(2+X,2+Y),2,3
2150 RETURN
2200 *** V **
2205 LINE(0+X,0+Y)-(20+X,60+Y)
2210 LINE-(30+X,60+Y)
2215 LINE-(50+X,0+Y)
2220 LINE-(40+X,0+Y)
2225 LINE-(25+X,50+Y)
2230 LINE-(10+X,0+Y)
2235 LINE-(0+X,0+Y)
2240 PAINT(3+X,3+Y),2,3
2245 RETURN
2300 *** W **
2305 LINE(0+X,0+Y)-(8+X,60+Y)

```

```

2310 LINE-(18+X,60+Y)
2315 LINE-(25+X,40+Y)
2320 LINE(50+X,0+Y)-(41+X,60+Y)
2325 LINE-(31+X,60+Y)
2330 LINE-(24+X,40+Y)
2335 LINE(0+X,0+Y)-(10+X,0+Y)
2340 LINE-(15+X,40+Y)
2345 LINE(50+X,0+Y)-(40+X,0+Y)
2350 LINE-(34+X,40+Y)
2355 LINE-(30+X,25+Y)
2360 LINE-(20+X,25+Y)
2365 LINE-(15+X,40+Y)
2370 PAINT(3+X,3+Y),2,3
2375 RETURN
2400 '** X **
2405 LINE(0+X,0+Y)-(20+X,30+Y)
2410 LINE-(0+X,60+Y)
2415 LINE-(10+X,60+Y)
2420 LINE-(25+X,37+Y)
2425 LINE-(40+X,60+Y)
2430 LINE-(50+X,60+Y)
2435 LINE-(31+X,30+Y)
2440 LINE-(50+X,0+Y)
2445 LINE-(40+X,0+Y)
2450 LINE-(25+X,22+Y)
2455 LINE-(10+X,0+Y)
2460 LINE-(0+X,0+Y)
2465 PAINT(3+X,3+Y),2,3
2470 RETURN
2500 '** Y **
2505 LINE(0+X,0+Y)-(20+X,30+Y)
2510 LINE-(20+X,60+Y)
2515 LINE-(31+X,60+Y)
2520 LINE-(31+X,30+Y)
2525 LINE-(50+X,0+Y)
2530 LINE-(40+X,0+Y)
2535 LINE-(25+X,22+Y)
2540 LINE-(10+X,0+Y)
2545 LINE-(0+X,0+Y)
2550 PAINT(3+X,3+Y),2,3
2555 RETURN
2600 '** Z **
2605 LINE(0+X,0+Y)-(50+X,0+Y)
2610 LINE-(50+X,10+Y)
2615 LINE-(10+X,50+Y)
2620 LINE-(50+X,50+Y)
2625 LINE-(50+X,60+Y)
2630 LINE-(0+X,60+Y)
2635 LINE-(0+X,48+Y)
2640 LINE-(38+X,10+Y)
2645 LINE-(0+X,10+Y)
2650 LINE-(0+X,0+Y)
2655 PAINT(2+X,2+Y),2,3
2660 RETURN
20020 'LINE(0+X,0+Y)-(50+X,60+Y),2,B

```

PROGRAMME 7.20

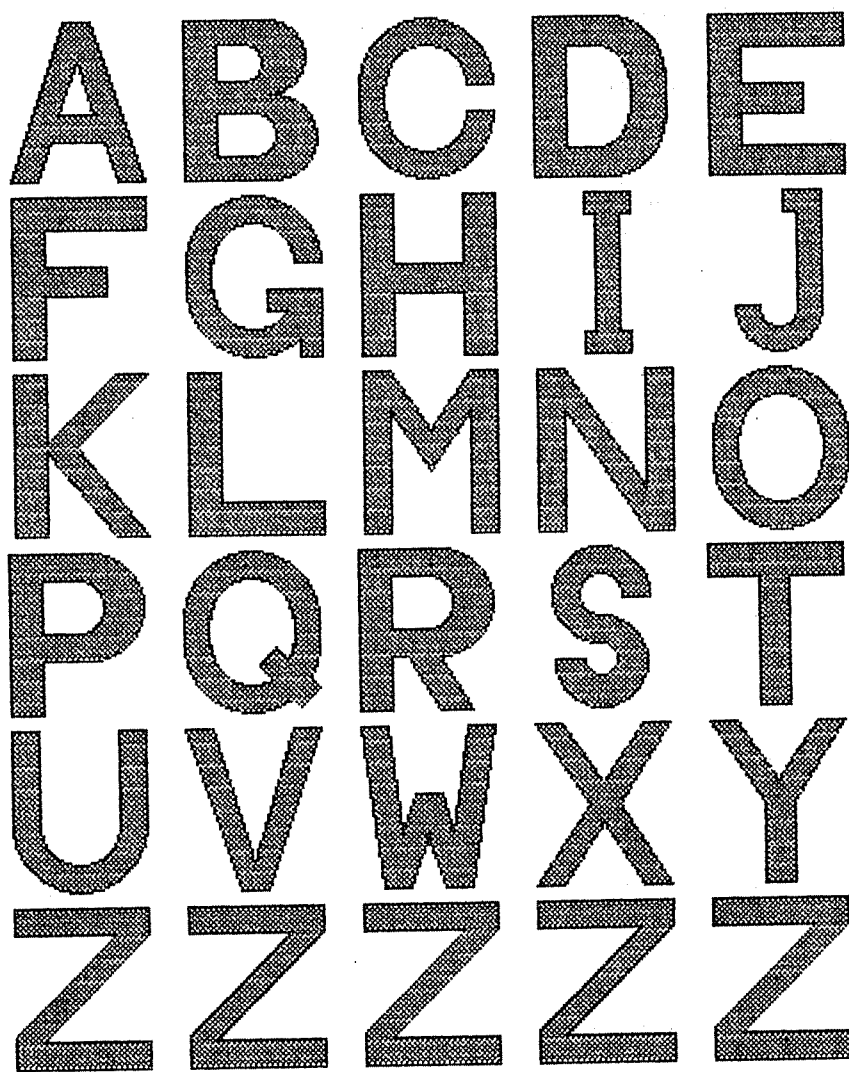


Fig. 7.17

7.19 L'ANNOTATION DES CAMEMBERTS

Aux paragraphes 3.2.5 et 4.6 nous avons étudié comment dessiner des camemberts, mais nous ne savons pas encore comment positionner les légendes qui identifient chaque morceau. Dans le programme 4.14 nous avons appris comment trouver les points équidistants des frontières de chaque morceau. Ces points étaient utilisés pour commencer le coloriage, mais conviendront également pour positionner les commentaires.

Vous vous rappelez qu'on utilisait le demi-rayon pour trouver les points se trouvant à mi-chemin entre le centre et la circonférence. Ici nous voulons que les commentaires soient un peu plus proches de la circonférence, aussi nous allons multiplier le rayon par 0,8. Aux lignes 60 à 90 du programme 7.21 les libellés et poids de chaque article sont rentrés dans les tableaux M et T\$. Aux lignes 100 à 240 le camembert est dessiné exactement comme dans le programme 4.14. Pour calculer les positions des étiquettes, nous utilisons la moyenne des angles qui déterminent la limite du morceau, le rayon multiplié par 0,8 et les équations d'un point en coordonnées polaires (puisqu'on a l'angle et le rayon).

On trouve les coordonnées de la cellule de caractère par la formule 7.2, mais il y a encore un calcul à faire : à la figure 7.18 on voit que les étiquettes qui tombent sur le côté gauche du cercle sont mieux disposées imprimées à partir de l'extérieur du cercle au contraire de celles de droite où les étiquettes sont imprimées de la manière habituelle. Pour déterminer sur lequel des deux côtés le titre se trouve, le programme vérifie chaque coordonnée X : si elle est inférieure à celle du centre du cercle, une chaîne de plusieurs CHR\$(29) ayant la même longueur que l'étiquette est imprimée ; ceci déplace le curseur sur la gauche de telle sorte que la chaîne imprimée ait son dernier caractère dans la cellule calculée.

```

0  '** 7-21 : Labeled pie **
10 SCREEN 1:CLS:DIM M(20),T$(20)
20 DEF FN ANG(X)=2*PI*X/100
30 PI=3.141593:A=0:F=1.745329E-02
40 CNTRX=160:CNTRY=100
50 RADIUS=98:ASPECT=5/6
60 INPUT"Number of items ";N
70 FOR I=1 TO N:
   PRINT"amount,title ";
74   INPUT M(I),T$(I)
76 NEXT
80 FOR I=1 TO N:
   T=T+M(I):
   NEXT
90 CLS
100 FOR I=1 TO N:
   M(I)=M(I)*100/T:
   NEXT
110 FOR I=1 TO N
120   V=FN ANG(M(I))
130   COL=I MOD 4
140   IF A+V>6.283186
      THEN
        A=6.283186-V
150   CIRCLE(CNTRX,CNTRY),RADIUS,COL,-A,-(A+V)
160   ANG=(A+A+V)/2
170   X=CNTRX+RADIUS*.5*COS(ANG)
180   Y=CNTRY+RADIUS*.5*SIN(ANG)*ASPECT
190   PAINT(X,200-Y),COL,COL
191   CIRCLE(CNTRX,CNTRY),RADIUS,3,-A,-(A+V)
192   A=A+V
194 NEXT
200 A=0

```

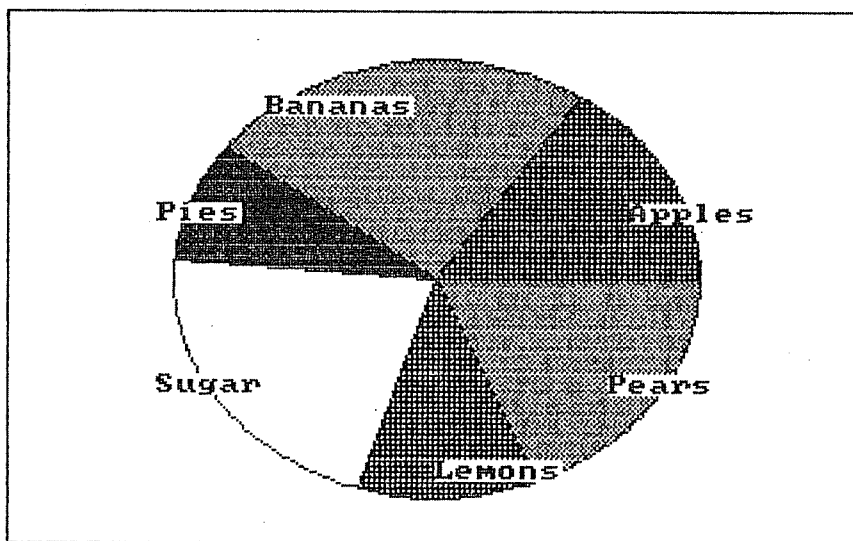


Fig. 7.18

```

410 FOR I=1 TO N
420   V=FN ANG(M(I))
440   IF A+V>6.283186
      THEN
        A=6.283186-V
460   ANG=(A+A+V)/2
470   X=CNTRX+RADIUS*.8*COS(ANG)
480   Y=CNTRY+RADIUS*.8*SIN(ANG)*ASPECT
490   LOCATE 1+(200-Y)/8,1+X/8:
      IF X<CNTRX
        THEN
          PRINT STRING$(LEN(T$(I)),29);
491   PRINT T$(I);
492   A=A+V
494 NEXT

```

PROGRAMME 7.21

EXERCICES

1. Les instructions
`LOCATE 1 + 24*RND,1 + 38*RND:PRINT CHR$(127*RND):`
 peuvent-elles provoquer une remontée de l'écran vers le haut ?
 Quand ?
2. Ecrire un sous-programme d'entrée en utilisant les fonctions CSRLIN
 et POS pour trouver la position d'origine du curseur.

3. Modifier le sous-programme d'entrée du programme 7.14 de telle sorte que X soit calculé avec la longueur de BUFF\$.
4. Ecrire un sous-programme d'entrée dans lequel chaque nouveau caractère soit imprimé d'une couleur différente.

NOTES

1. Nous appelons « caractères standards » ceux qui sont affichés par la plupart des ordinateurs et des terminaux, de CHR\$(0) à CHR\$(127), une gamme qui comprend toutes les lettres majuscules et minuscules, les chiffres et les symboles, plus l'ensemble des caractères de contrôle. Dans le mode texte, l'IBM PC peut afficher beaucoup plus de caractères.
2. Le symbole « \ » signifie division entière, ce qui revient à prendre la partie entière d'une division.
3. Notez qu'ici l'ordre des arguments est renversé.
4. Dans le mode texte, quand le curseur est allumé, il clignote automatiquement.
5. La souplesse de l'instruction INPUT va de pair avec une perte de contrôle : l'écran en entier peut être effacé par l'utilisateur (ctrl-HOME), la ligne peut être effacée (ESC), le curseur peut être déplacé, n'importe quel nombre de caractères peut être tapé, etc.
6. Pour une étude approfondie des entrées et de toutes leurs possibilités, voir l'ouvrage FANCY PROGRAMMING WITH IBM PC BASIC, du même auteur.
7. Ça peut être un caractère de contrôle ou un caractère dans la gamme CHR\$(128)..CHR\$(255).

8

Les transformations

Dans ce chapitre, nous décrivons plusieurs techniques pour manipuler les images : la translation qui est le déplacement d'une image à un emplacement différent sur l'écran ; les rotations, utilisées pour changer l'orientation angulaire ; le changement d'échelle utilisé pour agrandir ou réduire une image ; les distorsions qui allongent, compriment ou font tourner des parties d'une figure ; et les manipulations telles que la production de l'image miroir d'un dessin. Nous montrons également plusieurs applications pratiques de ces techniques, comme produire des ellipses orientées sous différents angles, et tout un nouvel ensemble de courbes reliées à celles du paragraphe 3.4.5.

Toutes les manipulations de ce chapitre sont faites sur des images à deux dimensions, c'est-à-dire planes. Au chapitre 11 nous étudierons des méthodes pour manipuler des formes tri-dimensionnelles.

8.1 LES TRANSLATIONS

Les translations ont pour objet le déplacement parallèle de figures sur l'écran. Il existe deux méthodes de translation : le décalage et la transformation des paramètres.

8.1.1. Le décalage

Pour décaler une figure (la déplacer à un emplacement différent), des valeurs de décalage sont ajoutées à chaque X et Y de telle sorte que

chaque point est repositionné. Supposez que la figure à décaler soit une ligne allant du point (0,0) au point (50,50). Nous voulons avoir le premier point à (130,70). Pour faire cela nous devons ajouter 130 aux deux X et 70 aux deux Y. L'instruction devient donc `LINE(0 + 130,0 + 70)-(50 + 130,50 + 70)`. Pour rendre cette opération de décalage plus générale, les facteurs de décalage X et Y doivent être des variables, non des constantes. Le programme 8.1 demande les coordonnées du premier point de la nouvelle ligne et la décale à son nouvel emplacement.

```
0  °° 8-1 : Moving a line °°
10 SCREEN 1:CLS
20 LOCATE 1,1:INPUT"New coordinates ";XI,YI
30 LINE(0+XI,0+YI)-(50+XI,50+YI)
40 GOTO 20
```

PROGRAMME 8.1

Les fonctions NX et NY définies à la ligne 25 du programme 8.2 ajoutent l'incrément à la variable correspondante. L'utilisation de fonctions rend en général les programmes plus clairs, et économise des variables, donc de la mémoire. On utilise les tableaux X et Y pour mémoriser les coordonnées des points de départ et d'arrivée. Le point de départ du premier point sera mémorisé dans X(1,1),Y(1,1) et le point d'arrivée sera mémorisé dans X(1,2),Y(1,2). Essayez le programme avec le carré dont les coordonnées sont classées dans la table 8.1.

```
0  °° 8-2 : Move vector figure °°
10 SCREEN 1:CLS
20 DIM X(100,2),Y(100,2):I=1
25 DEF FN NX(X)=X+XI:DEF FN NY(Y)=Y+YI
30 PRINT"Line ";I;" starting point (X) ";
40 INPUT W$:
   IF W$=""
     THEN
       80
50 X(I,1)=VAL(W$):INPUT"Starting point (Y) ";Y(I,1)
60 INPUT"Ending point (X) ";X(I,2)
65 INPUT"Ending point (Y) ";Y(I,2)
70 I=I+1:GOTO 30
80 CLS:NP=I-1
90 LOCATE 1,1:INPUT"New starting point (X,Y) ";XI,YI
100 FOR I=1 TO NP
110  LINE(FN NX(X(I,1)),FN NY(Y(I,1)))-
      (FN NX(X(I,2)),FN NY(Y(I,2)))
120 NEXT
130 GOTO 90
```

PROGRAMME 8.2

Table 8.1.

ORIGINE		DESTINATION	
X	Y	X	Y
0	0	40	0
40	0	40	40
40	40	0	40
0	40	0	0

8.1.2 La transformation des paramètres

Transformer c'est prendre une valeur et la convertir en une autre, suivant un mode ou une formule préétabli. Le but de cette technique est de faciliter le choix des paramètres, de sorte que les nombres utilisés correspondent directement aux valeurs de l'écran. Prenons un exemple : nous voulons tracer la fonction $F(X) = X/100$ en partant de $X = -160$. Nous pourrions utiliser une boucle FOR pour traverser l'écran en partant de -160 mais alors nous devrions calculer le point de terminaison (qui n'est pas 319). Une meilleure technique consiste à utiliser les valeurs habituelles de début et de fin pour la boucle FOR (0 et 319), et transformer chaque valeur de sorte que la fonction soit calculée comme débutant à -160 . Si nous définissons la fonction comme $FN\ TR(X) = X - 160$, nous pouvons alors utiliser les valeurs de la boucle FOR directement en les « filtrant » à travers cette fonction, comme le montre le programme 8.3.

```

0  ° 8-3 : Function with mapped parameters °
10 SCREEN 1:CLS
20 DEF FN S(X)=X^2/100
30 DEF FN TR(X)=X-160
40 FOR X=0 TO 319
50   Y=180-FN S(FN TR(X))
60   PSET(X,Y)
80 NEXT

```

PROGRAMME 8.3

Une utilisation plus élaborée de la transformation est expliquée au paragraphe 2.5.2.

8.2 L'INCLINAISON

Si le programme manipule les coordonnées des points ou des blocs avant qu'ils ne soient dessinés, il est possible d'avoir une copie de la partie originale avec une orientation différente. Le programme 8.4 ajoute

la valeur de I à la coordonnée verticale du point objet. La reproduction est la copie inclinée de l'original.

```
0 *** 8-4 : Slanting **
5 CLS:INPUT"X,Y ";X,Y
10 SCREEN 1:COLOR 1,0:CLS
20 FOR I=0 TO 20 STEP 3
30   LINE(3,I)-(20,I)
40 NEXT
41 LINE(0,0)-(319,199),3,B
45 LOCATE 2,4:PRINT"Gargolae"
50 FOR I=0 TO 90
60   FOR J=0 TO 20
70     PSET(X+I,Y+J+I),POINT(I,J)
80   NEXT
90 NEXT
```

PROGRAMME 8.4

On peut contrôler l'effet d'inclinaison si, au lieu de I (qui fait descendre la figure de 1 pixel à chaque incrémentation horizontale de 1), on ajoute seulement une fraction de I. Si par exemple on ajoute $I/2$ à J, la figure descendra de 1 pixel pour chaque incrémentation de deux pixels horizontalement. La figure 8.1 a été dessinée en étendant le programme 8.4 dans lequel trois différents facteurs d'inclinaison ont été utilisés : I, I^* ,5 et I^* ,2.

Le programme 8.5 demande le facteur d'inclinaison en ligne 30. Ce nombre, multiplié par I, sera ajouté à J (qui à son tour a été ajouté à Y pour tracer la copie dans le sens voulu).

```
0 *** 8-5 : Variable slanting **
10 SCREEN 1:CLS
20 X=0:Y=20
30 INPUT"Slanting factor ";SF
40 CLS
50 LINE INPUT W$
60 FOR I=0 TO 159
70   FOR J=0 TO 7
80     PSET(X+I,Y+J+I/SF),POINT(I,J)
90   NEXT
95 NEXT
```

PROGRAMME 8.5

Dans le programme 8.6 on a ajouté la valeur de I aux coordonnées verticales, et la valeur J a été soustraite de la coordonnée horizontale. Ceci

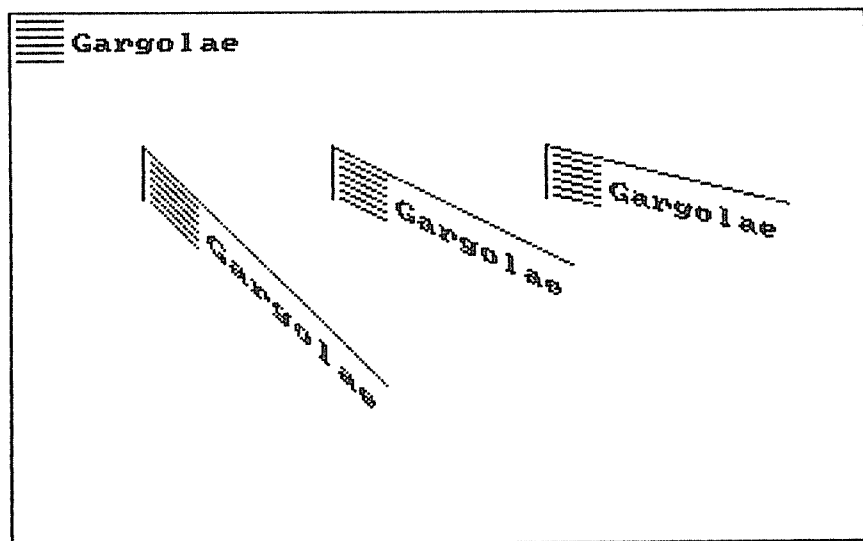


Fig. 8.1

inclina à la fois les composantes X et Y de chaque point, le résultat étant une rotation de -45 degrés de la figure, produite par les lignes 40 à 60. La figure 8.2 montre la rotation accomplie avec cette méthode.

```

0  "" 8-6 : Dual slanting ""
10 SCREEN 1:CLS
20 INPUT "New origin (X,Y) ";X,Y
30 CLS
40 FOR I=0 TO 30 STEP 3
50   LINE (0,15)-(30,I)
60 NEXT
70 FOR I=0 TO 30
80   FOR J=0 TO 30
90     S=POINT(I,J)
100    X1=X+I:Y1=Y+J
110    PSET (X1-J,Y1+I),S
120  NEXT
130 NEXT

```

PROGRAMME 8.6

8.3 LA ROTATION

Bien qu'une figure puisse être placée n'importe où sur l'écran avec les méthodes de translation expliquées au paragraphe précédent, une telle

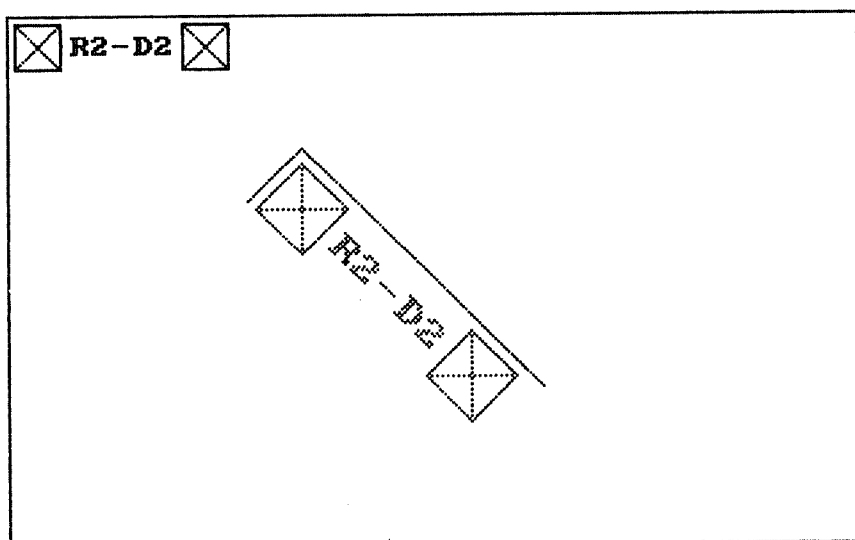


Fig. 8.2

figure aura toujours la même orientation, à savoir -45 degrés. On peut généraliser facilement la méthode pour réaliser des rotations de $45, 135$ et 225 degrés¹, mais la plupart des autres angles sont difficiles à obtenir. Au chapitre 6 nous avons vu que l'instruction DRAW permet une forme limitée de rotation, toutefois ces rotations ne peuvent être que d'angles qui sont multiples de 90 degrés, et pour des figures créées par le sous-langage DRAW. Au chapitre 7 nous avons vu une autre méthode pour changer l'orientation des figures, dans ce cas des caractères.

On peut effectuer une rotation plus fine, sans limitation des angles, en utilisant le changement de coordonnées habituellement employé en trigonométrie. La figure 8.3 montre le système de coordonnées X, Y habituel, et une seconde paire d'axes X' et Y' (en pointillé). Il y a un angle THETA entre les axes X et X' (et donc entre les axes Y et Y'). Un point de coordonnées X, Y peut être repéré dans le nouveau système de coordonnées en appliquant simplement les équations 8.1 et 8.2.

$$X1 = \cos(\text{THETA}) * X + \sin(\text{THETA}) * Y \quad (8.1)$$

$$Y1 = -\sin(\text{THETA}) * X + \cos(\text{THETA}) * Y \quad (8.2)$$

Par exemple, pour trouver les coordonnées du point $(3, 5)$ dans un système de coordonnées faisant un angle de 45 degrés par rapport au système $X-Y$ nous effectuons les deux opérations :

$$X1 = \cos(45) * 3 + \sin(45) * 5 = 5.830484$$

$$Y1 = -\sin(45) * 3 + \cos(45) * 5 = 7.389951E-02$$

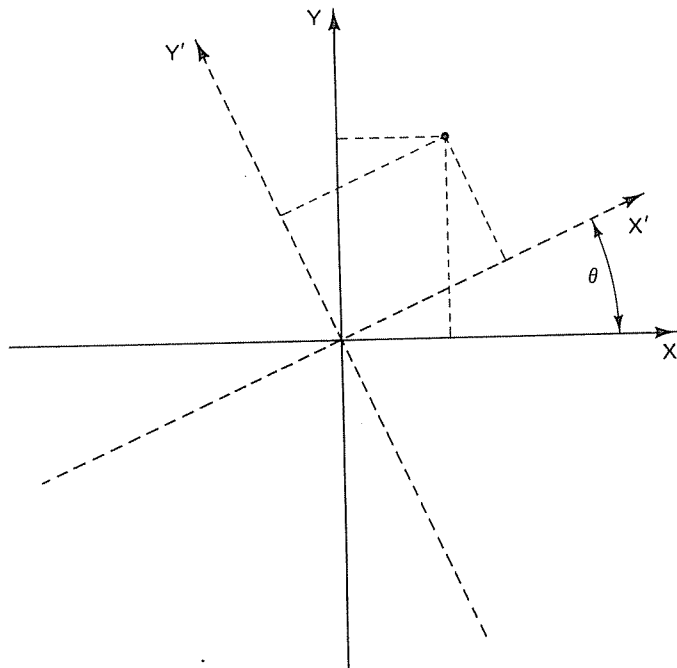


Fig. 8.3

Le programme 8.7 utilise les équations 8.1 et 8.2 pour repositionner et faire tourner le dessin produit par les lignes 50 à 70.

```

0  *** 8-7 : Rotation **
10 SCREEN 1:CLS
20 INPUT"Angle ";THETA
30 INPUT"New position (X,Y) ";NX,NY
40 CLS
50 FOR I=0 TO 15 STEP 4
60   LINE(I,I)-(30-I,30-I),3,B
65   LINE(I,I)-(30-I,I),2
70 NEXT
80 THETA=THETA*1.745329E-02
90 CS=COS(THETA):SN=SIN(THETA)
100 FOR I=0 TO 30
110   FOR J=0 TO 30
120     X=CS*I+SN*J:Y=-SN*I+CS*J
130     PSET(X+NX,Y+NY),POINT(I,J)
140   NEXT
150 NEXT

```

PROGRAMME 8.7

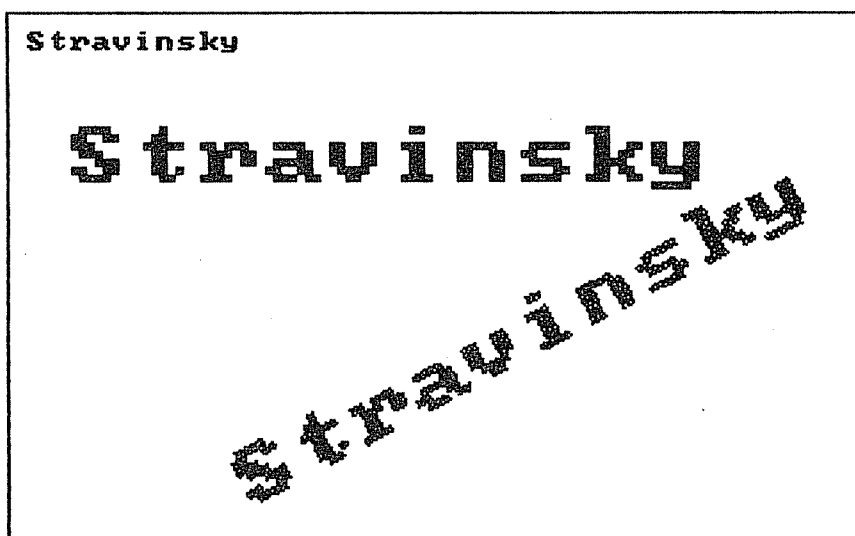


Fig. 8.4

La figure 8.4 a été produite d'abord en agrandissant la chaîne « Stravinsky » et ensuite en la faisant tourner de 20 degrés.

Le programme 8.8 montre la translation et la rotation répétée d'une ligne. Les points limites d'origine sont (0,-80) et (0,-5), c'est-à-dire que la ligne est complètement en dehors de l'écran. Pour calculer les points ayant tourné nous utilisons les équations définies aux lignes 10 et 15. Notez que cette définition fonctionnelle ajoute de la souplesse en permettant le calcul de rotations sans utiliser de variables intermédiaires. L'angle THETA est multiplié par F pour faciliter les calculs en degrés. Après que chaque point eut tourné on ajoute les incréments (160,100) à ses coordonnées de telle sorte que le centre du système soit en (160,100) et les lignes tournent autour de ce point. La figure 8.5 montre le graphique résultant.

```
0  "" 8-8 : Rotating line ""
10 DEF FN NX(X,Y,A)=COS(A)*X+SIN(A)*Y
15 DEF FN NY(X,Y,A)=-SIN(A)*X+COS(A)*Y
20 F=1.745329E-02
30 SCREEN 1:CLS
40 FOR THETA=0 TO 360 STEP 36
50   TEMP=THETA*F
60   LINE(FNNX(0,-80,TEMP)+160,FNNY(0,-80,TEMP)+100)-
      (FNNX(0,-5,TEMP)+160,FNNY(0,-5,TEMP)+100)
70 NEXT
```

PROGRAMME 8.8

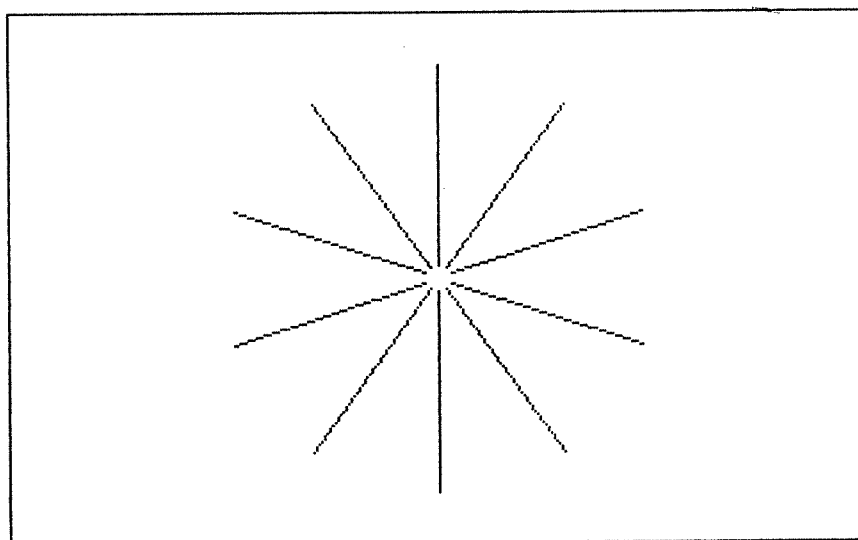


Fig. 8.5

8.3.1 Cadres et blocs inclinés

Les cadres et les blocs que nous avons étudiés au chapitre 2 étaient limités à des lignes parallèles aux axes X et Y. Nous pouvons maintenant faire tourner un cadre en utilisant les équations 8.1 et 8.2 et lui donner n'importe quelle orientation. Le programme 8.9 demande un angle à la ligne 30 et trace le bloc qui était déterminé à l'origine par les points $(-70,-40)$ et $(70,40)$, déplaçant le centre (le point $(0,0)$) au centre de l'écran. Notez que nous avons inclus la translation, de même que la rotation, dans les fonctions des lignes 10 et 15.

```

0  °° 8-9 : Rotated block °°
10 DEF FN NX(X,Y,A)=160+(COS(A)*X+SIN(A)*Y)
15 DEF FN NY(X,Y,A)=100+(-SIN(A)*X+COS(A)*Y)
20 CLS
30 INPUT"Angle ";THETA
40 THETA=THETA*1.745329E-02
50 SCREEN 1:CLS
60 LINE(FNNX(-70,-40,THETA),FNNY(-70,-40,THETA))-
    (FNNX(70,-40,THETA),FNNY(70,-40,THETA)),2
70 LINE(FNNX(70,-40,THETA),FNNY(70,-40,THETA))-
    (FNNX(70,40,THETA),FNNY(70,40,THETA)),2
80 LINE(FNNX(70,40,THETA),FNNY(70,40,THETA))-
    (FNNX(-70,40,THETA),FNNY(-70,40,THETA)),2
90 LINE(FNNX(-70,40,THETA),FNNY(-70,40,THETA))-

```

```

(FNNX(-70,-40,THETA),FNNY(-70,-40,THETA)),2
100 PAINT(160,100),2,2

```

PROGRAMME 8.9

La figure 8.6 montre le bloc produit par une rotation de 75 degrés.

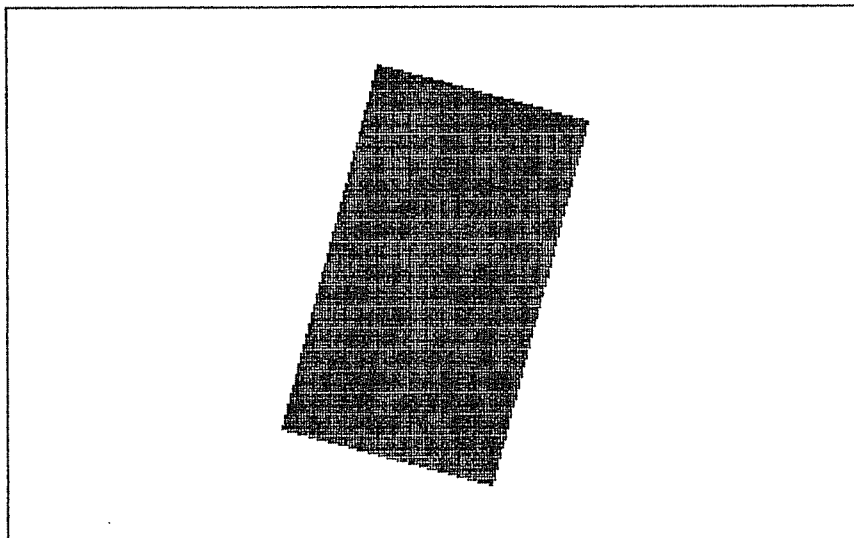


Fig. 8.6

8.3.2 Ellipses d'inclinaison quelconque

Les ellipses étudiées au chapitre 3 ne pouvaient être orientées que parallèlement à l'axe des X ou à l'axe des Y. Avec la méthode de tracé en coordonnées polaires, et de rotation expliquée au paragraphe précédent, il est possible d'obtenir des ellipses à orientation d'axes quelconque. Le programme 8.10 demande en ligne 30 l'angle désiré et trace l'ellipse résultante avec un grand axe de 110 et un petit de 55. La figure 8.7 montre l'ellipse résultante avec un angle d'inclinaison de -45 degrés.

```

0  °° 8-10 : Rotated ellipse °°
10 SCREEN 1:CLS
20 F=1.745329E-02
30 INPUT"Angle ";THETA:THETA=THETA*F
40 CLS
50 CS=COS(THETA):SN=SIN(THETA)
60 RADIUS=55
70 FOR A=0 TO 360

```

```

80  X=RADIUS*COS(A*F)
90  Y=2*RADIUS*SIN(A*F)
100 X1=X*CS+Y*SN
110 Y1=-SN*X+CS*Y
120 IF A=0
      THEN
        PSET(160+X1,100+Y1)
      ELSE
        LINE-(160+X1,100+Y1)
130 NEXT

```

PROGRAMME 8.10

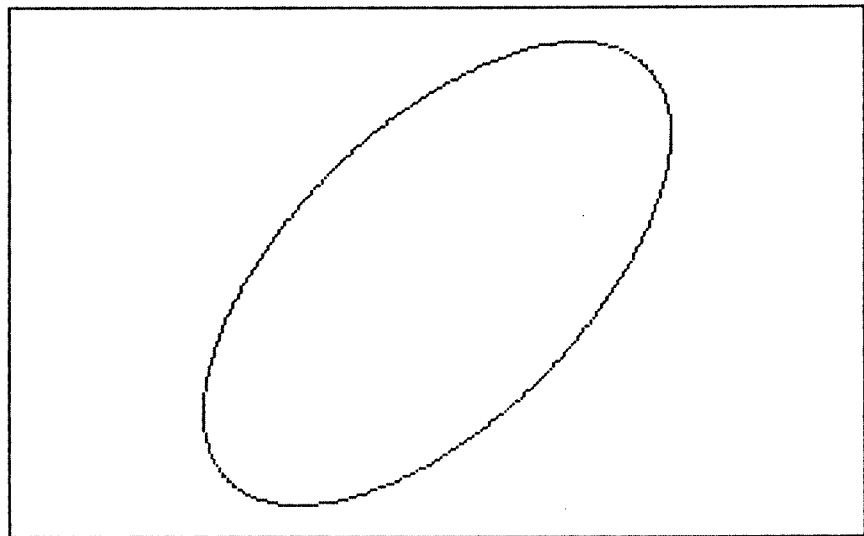


Fig. 8.7

8.4 LES SPIRALES

Si une figure est dessinée plusieurs fois et que chaque fois on la fasse tourner d'un angle qui est progressivement incrémenté, le résultat sera semblable aux figures tracées au paragraphe 3.4.5, avec l'avantage qu'ici il n'y aura pas de limites quant à la forme de l'objet. Par exemple, le programme 8.11 trace une ellipse en utilisant les coordonnées polaires. Quand l'angle de l'ellipse croît, on y ajoute une petite valeur, de telle sorte que quand la première ellipse est terminée, elle est légèrement inclinée par rapport à son point de départ. Le processus est répété plusieurs fois, jusqu'à ce qu'un des points terminaux de l'ellipse coïncide avec le premier point tracé.

```

0  ' ** 8-11 : Rotating ellipse **
10 SCREEN 1:CLS
15 START=0:ENDING=80
20 R=60
30 FOR A=START TO ENDING STEP .05
40   X=R*COS(A)
50   Y=1.5*R*SIN(A)
52   X1=X*COS(B)-Y*SIN(B)
54   Y1=X*SIN(B)+Y*COS(B)
60   IF A=START
       THEN
           PSET(160+X1,100+Y1)
       ELSE
           LINE-(160+X1,100+Y1)
70   B=B+.002
80 NEXT

```

PROGRAMME 8.11

La figure 8.8 montre le graphique réalisé.

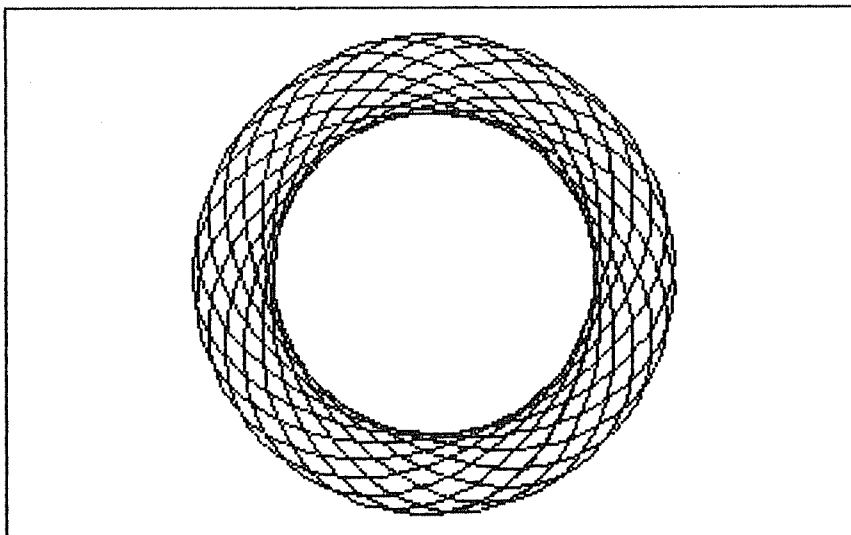


Fig. 8.8

Le programme 8.12 applique la même technique à un polygone, en réduisant le rayon quand l'angle augmente. Les figures 8.9, 8.10 et 8.11 montrent les graphiques ainsi obtenus à partir des polygones à 3, 4 et 8 côtés.

```

0  ' ** 8-12 : Rotating polygon **
10 SCREEN 1:CLS
15 INPUT "Number of sides ";S:S=360/S
20 INPUT "radius ";R
25 CLS:LINE(0,0)-(319,199),3,B:A=0

```

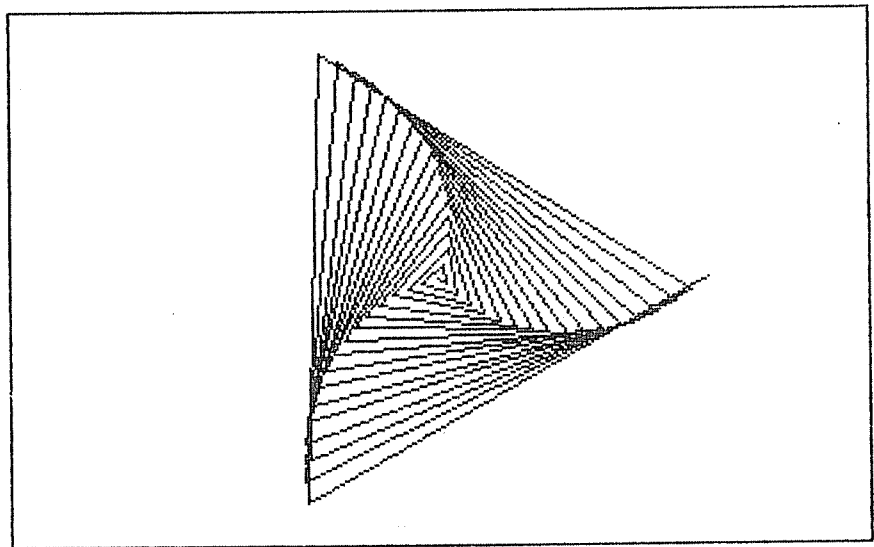


Fig. 8.9

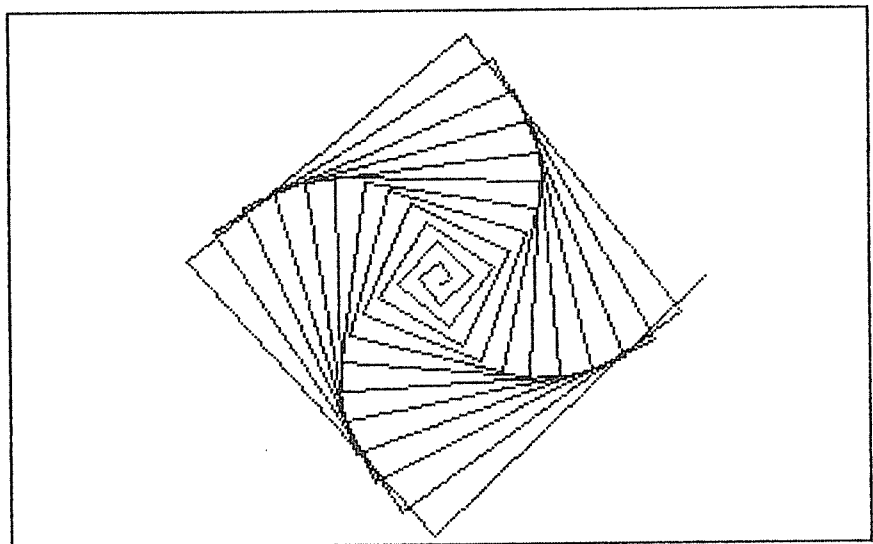


Fig. 8.10

```

30 WHILE R>0
40   X=R*COS((A+F)*1.745329E-02)
50   Y=R*SIN((A+F)*1.745329E-02)
60   IF A=0
       THEN
         PSET(160+X,100+Y)
       ELSE
         LINE-(160+X,100+Y)

```

```

70 R=R-1.2:F=F+1:A=A+S
80 WEND

```

PROGRAMME 8.12

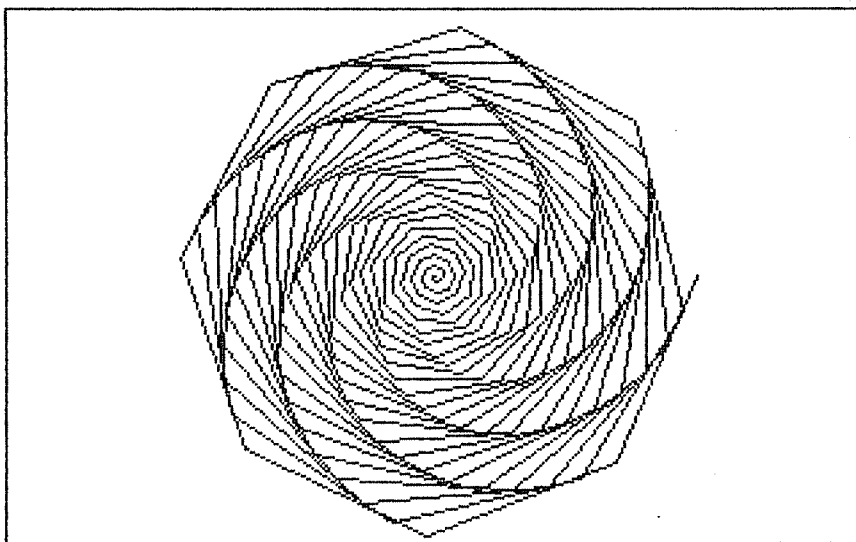


Fig. 8.11

8.5 TRANSLATIONS NON STANDARD

Les méthodes utilisées pour changer l'orientation et la direction des caractères au paragraphe 7.12 peuvent être utilisées pour produire toute une variété de translations non standard, telles que des images miroirs par rapport à l'axe des X ou des Y ou des inversions. Le seul changement à faire pour utiliser les programmes des paragraphes 7.12.1 à 7.12.7 pour les graphiques est dans la dimension de la zone balayée. Par exemple le programme 8.13 produit une image miroir de la partie gauche de l'écran, comme le montre la figure 8.12.

```

0  °° 8-13 : Lateral mirror °°
5  DEFINT A-Z
10  CLS:INPUT N:RANDOMIZE N
20  SCREEN 1:CLS
30  FOR I=1 TO 20
40    X=160°RND:Y=199°RND:COL=1+3°RND:
      RADIUS=30°RND
50    CIRCLE(X,Y),RADIUS,COL-1,,,1

```

```

60  PAINT(X,Y),COL-1,COL-1
70  CIRCLE(X,Y),RADIUS,COL,,,1
80  PAINT(X,Y),COL,COL
90  NEXT
100 FOR I=1 TO 20
110  X1=160°RND:Y1=199°RND:X2=X1+60°RND:
      Y2=Y1+50°RND:COL=4°RND
120  LINE(X1,Y1)-(X2,Y2),COL,BF
130  NEXT
140  ' Create mirror image
150 FOR I=0 TO 159
160  FOR J=0 TO 199
170  PSET(319-I,J),POINT(I,J)
180  NEXT
190 NEXT

```

PROGRAMME 8.13

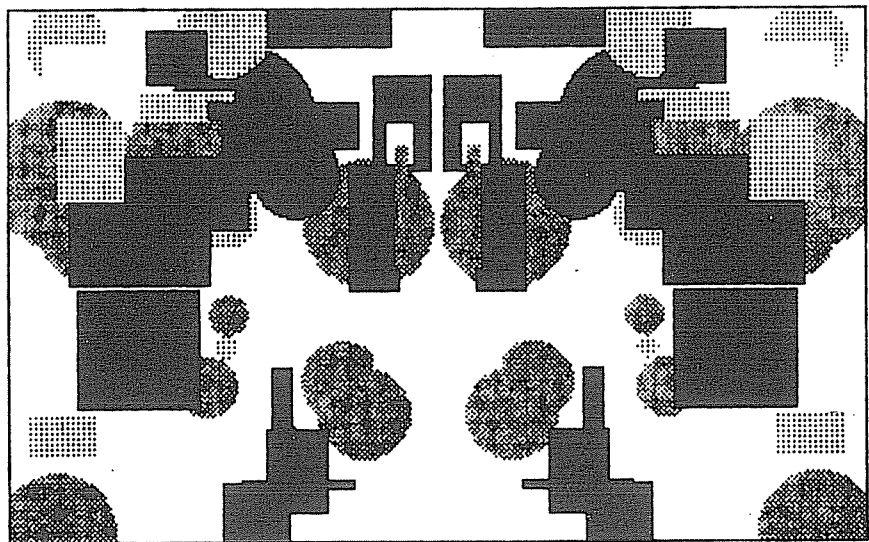


Fig. 8.12

Une autre famille de figures intéressantes peut être créée quand l'image est reflétée à la fois verticalement et horizontalement. La section de programme 8.14 utilise la figure créée par les lignes 5 à 130 du programme 8.13 et réalise l'effet kaléidoscopique montré en figure 8.13.

```

140  ' Create mirror image
150 FOR I=0 TO 159
160  FOR J=0 TO 100

```



```

170   PSET(319-I,J),POINT(I,J)
180   PSET(I,199-I),POINT(I,J)
190   PSET(319-I,199-I),POINT(I,J)
200   NEXT
210 NEXT

```

PROGRAMME 8.14

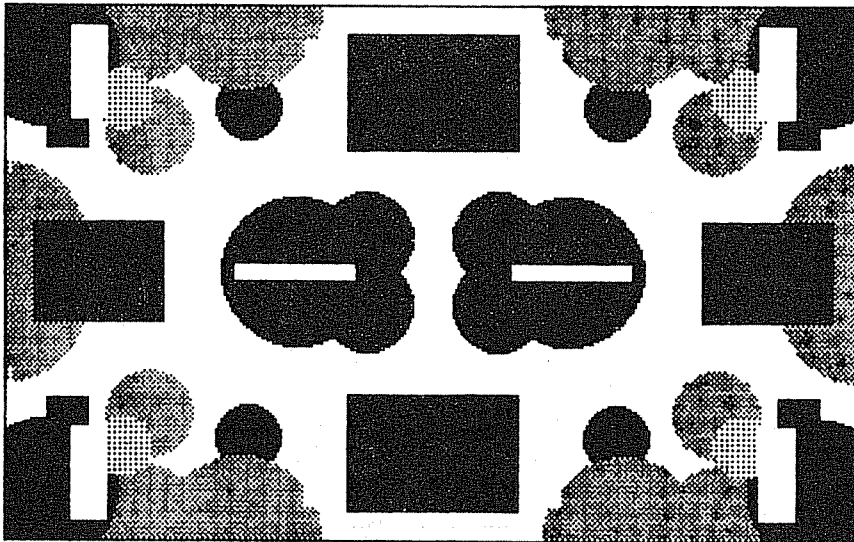


Fig. 8.13

8.6 LE CHANGEMENT D'ECHELLE

La méthode pour changer l'échelle des caractères présentée au paragraphe 7.11 peut être facilement adaptée pour changer l'échelle de n'importe quel type de graphique. En changeant les limites de la zone balayée au programme 7.10 (lignes 40 à 50), n'importe quelle portion de l'écran peut être agrandie ou réduite. (Quand HS et VS sont inférieurs à 1, la figure est réduite.) Le programme 8.15 agrandit ou réduit la figure produite aux lignes 30 à 50, selon la valeur affectée à SCALE à la ligne 10. Notez qu'ici nous n'utilisons qu'une seule variable pour les échelles verticales et horizontales parce que nous voulons que la figure garde ses proportions originales.

La figure 8.14 montre l'agrandissement effectué par le programme 8.15 avec SCALE=2,5, et la figure 8.15 le résultat avec SCALE=5.

```

0  °° 8-15 : Enlarging circles °°
10 X=71:Y=10:SCALE=2.5

```

```

20 SCREEN 1:CLS
30 FOR RADIUS=1 TO 35 STEP 3
40   CIRCLE(35,35),RADIUS,3,,,1
50 NEXT
60 FOR I=0 TO 70
70   FOR J=0 TO 70
80     S=POINT(I,J)
90     X1=X+I*SCALE:Y1=Y+J*SCALE
100    LINE(X1,Y1)-(X1+(SCALE-1),Y1+(SCALE-1)),S,BF
110   NEXT
120 NEXT

```

PROGRAMME 8.15

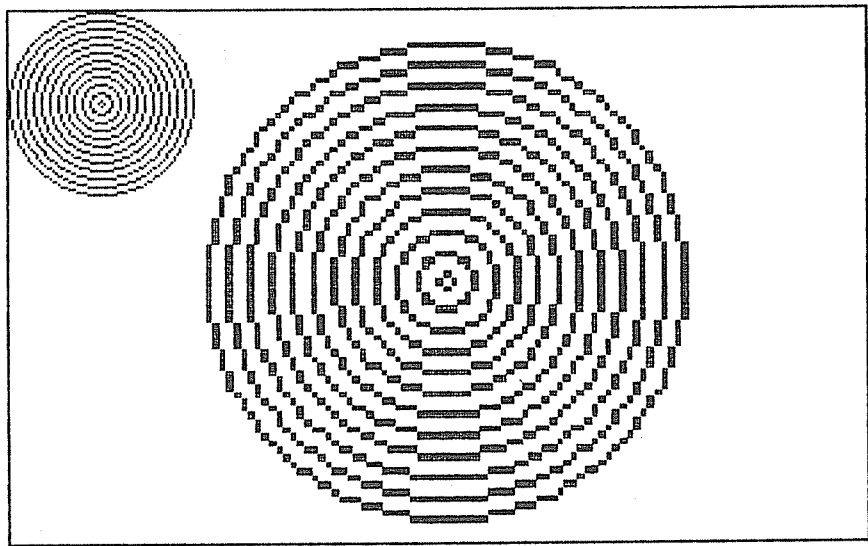


Fig. 8.14

8.7 LES DISTORSIONS

Nous allons décrire ici quatre types de distorsion, toutes étant des extensions de la méthode de changement d'échelle décrite au paragraphe 7.11.

8.7.1 Distorsion horizontale

Si le facteur d'échelle vertical utilisé au programme 7.10 est 1, et que l'on utilise un facteur d'échelle horizontale supérieur à 1, la figure origi-

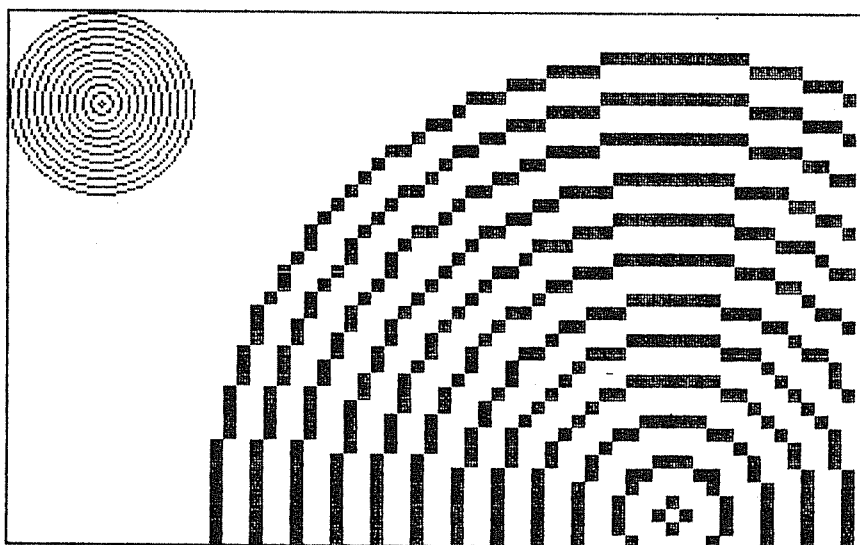


Fig. 8.15

nale sera étirée suivant l'axe des X, comme si elle était élastique. La figure 8.16 montre un exemple de ce type de distorsion.

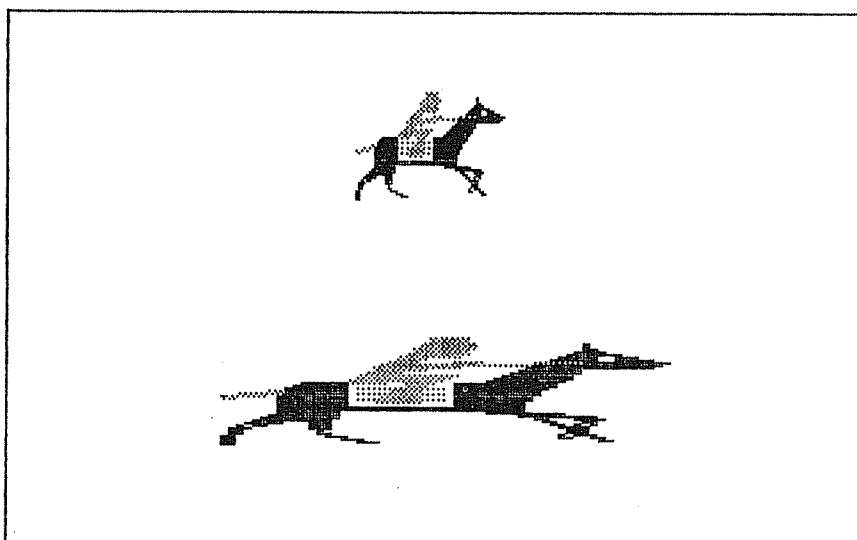


Fig. 8.16

8.7.2 Distorsion verticale

Ici les facteurs d'échelle sont inversés : 1 est utilisé comme facteur horizontal et le facteur vertical est supérieur à 1. La figure 8.17 montre un exemple de distorsion verticale.

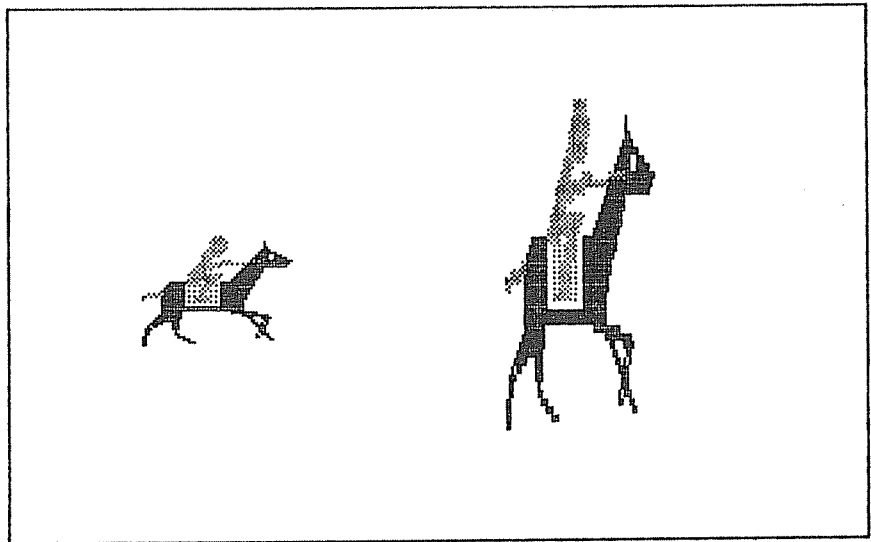


Fig. 8.17

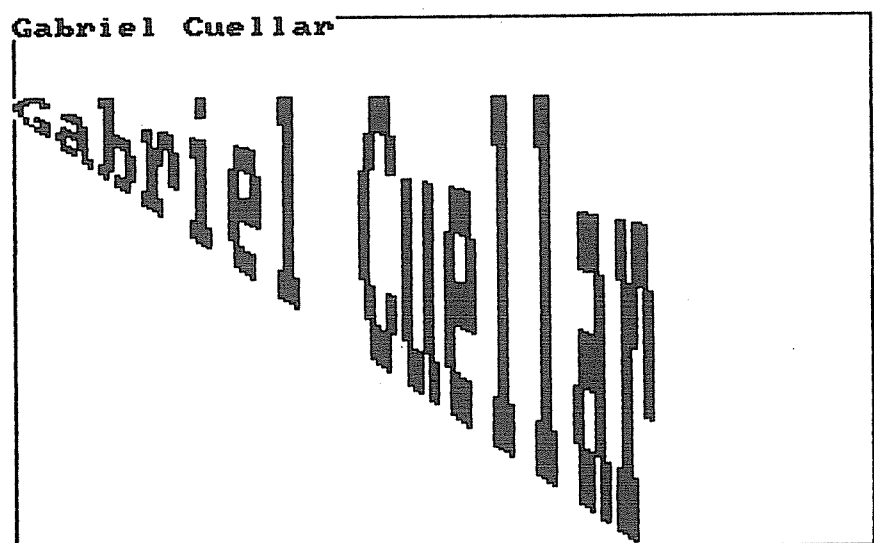


Fig. 8.18

8.7.3 Distorsion horizontale ou verticale progressive

Si le facteur d'échelle dans l'une des deux directions est progressivement agrandi (ou réduit), la forme sera déformée progressivement. La figure 8.18 montre un exemple de ce type de distorsion.

8.7.4 Distorsion progressive suivant les deux axes

Si les facteurs d'échelle vertical et horizontal sont progressivement réduits (ou agrandis), la distorsion ressemble à de la perspective. La figure 8.19 montre la chaîne « MILITA » déformée par des facteurs d'échelle décroissants.

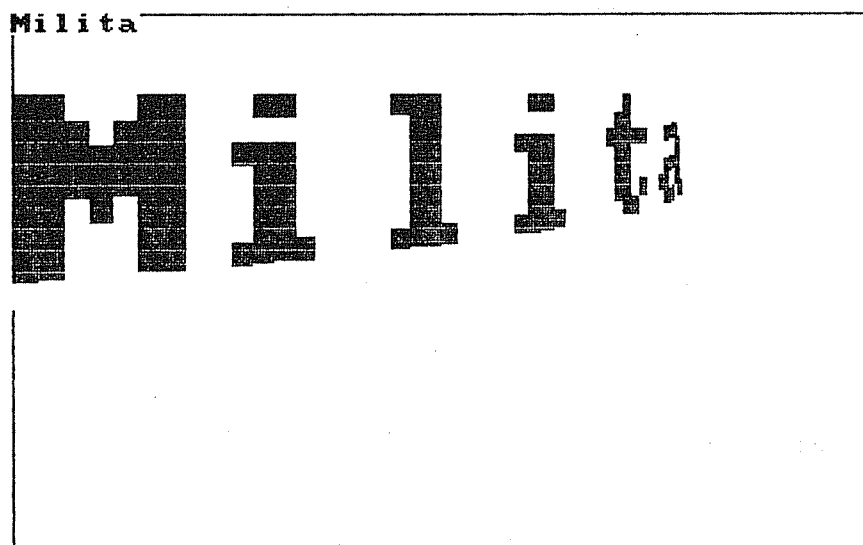


Fig. 8.19

EXERCICES

1. Ecrire deux versions d'un programme dans lequel une partie de l'écran soit tournée de 45 degrés. Dans l'une des versions utiliser la méthode du programme 8.6 et dans l'autre utiliser les équations 8.1 et 8.2.
2. Ecrire un programme dans lequel une partie de l'écran soit balayée et copiée au moyen d'un décalage progressif.
3. Ecrire un programme qui dessine la figure 8.20 (Conseil : comparez-le avec la figure 8.9).

4. Ecrire un programme qui utilise le principe de l'exercice 3 et remplisse l'écran avec des carrés comme ceux de la figure 8.10. Le graphique produit est très joli.
5. Si vous avez pris le temps de répondre correctement aux exercices 3 et 4, la figure 8.21 sera facile à comprendre. Ecrire un programme qui la génère.

NOTES

1. Cette méthode sera étendue pour produire des rotations multiples de 90 degrés, de même que pour produire des images-miroirs.

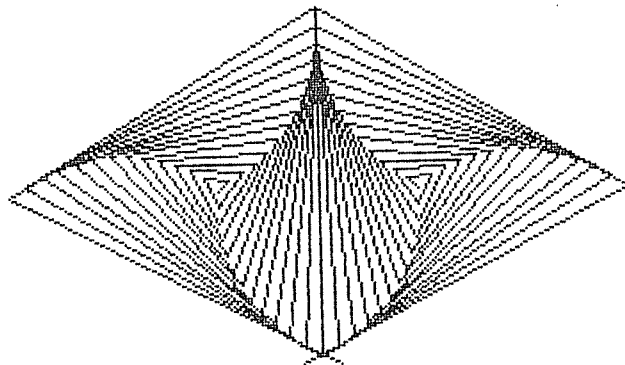


Fig. 8.20

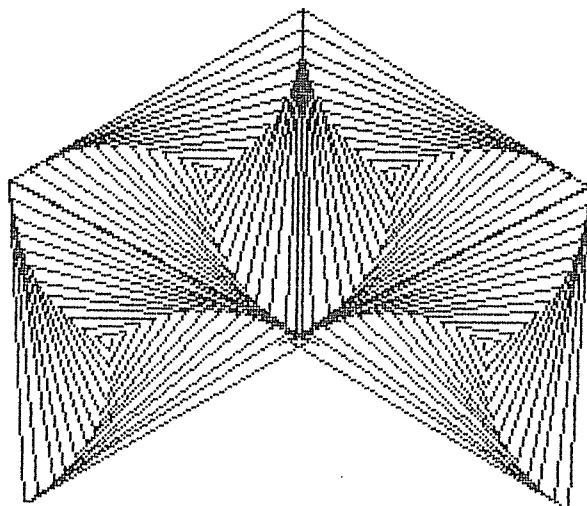


Fig. 8.21



9

Le transfert de parties de l'écran

L'instruction DRAW permet une grande souplesse parce que la plupart de ses instructions se réfèrent au dernier point envisagé (LRP) : pour affecter un nouvel emplacement à une figure, il suffit de changer le LRP. Quand une figure est créée par des instructions différentes de DRAW, la déplacer peut être long et compliqué. Dans ce chapitre nous allons étudier différentes méthodes pour mémoriser temporairement l'information concernant une image de telle sorte que l'on puisse la relocaliser et la traiter pour réaliser des effets spéciaux.

9.1 GET ET PUT

Il existe un couple d'instructions spécialement dédié au déplacement de zones entières de l'écran : GET et PUT. Voici une brève explication du processus :

1. Un tableau numérique est DIMensionné. Ceci réserve une zone en mémoire.
2. Une zone rectangulaire choisie de l'écran est copiée dans le tableau au moyen de GET. Après cela le tableau contient toutes les informations nécessaires pour reproduire la zone graphique.
3. L'information du tableau est recopiée sur l'écran au moyen de PUT. La zone sur laquelle on écrit peut être différente de celle sur laquelle l'information a été lue. En outre, l'information peut être traité de différentes manières pour obtenir des effets spéciaux.

Nous étudierons deux exemples avant d'expliquer les instructions GET et PUT en détail. Les lignes 20 à 80 du programme 9.1 dessinent une figure formée par une grille et des cercles concentriques.

```
0  ** 9-1 : Grid and circles **
5  DIM M(1000)
10 SCREEN 1:COLOR 0,0:CLS
20 FOR I=5 TO 50 STEP 5
30   CIRCLE (50,50),I,3,,,1
40 NEXT
50 FOR I=0 TO 100 STEP 5
60   LINE (I,0)-(I,100),2
70   LINE (0,I)-(100,I),2
80 NEXT
```

PROGRAMME 9.1

Il est possible de reproduire ce dessin en différents endroits de l'écran en changeant tous les paramètres de ligne et de colonne. Pour déplacer la figure suivant l'axe horizontal, il faut ajouter une variable à chaque valeur de colonne, et pour la déplacer suivant l'axe vertical il faut ajouter une variable à chaque valeur de ligne. Par exemple, on peut changer la ligne 30 en :

```
30   CIRCLE (X+50,Y+50),I,3,,,1.
```

Ce processus présente plusieurs inconvénients : ajouter des variables à chaque coordonnée est fastidieux, et le temps passé peut être trop long. Imaginez un jeu électronique où chaque fois qu'un bateau se déplace, le programme commence à dessiner des rectangles, des cercles et des lignes et à colorier différentes parties de couleurs différentes. Le temps que le processus soit fini, le joueur se sera endormi !

On peut ajouter le programme 9.2 au programme 9.1 pour faire en sorte que la reproduction de la figure soit un processus très rapide et très simple. Après que la figure ait été terminée, la ligne 90 lit (GET) la zone qui est dans le cadre (déterminée par les points (0,0) et (100,100) ¹ dans le tableau M). La ligne 100 efface l'écran et la ligne 110 restitue (PUT) la figure dans un nouvel emplacement choisi au hasard. Après un bref délai, la figure est placée dans un nouvel emplacement, et le processus est répété indéfiniment.

```
90 GET (0,0)-(100,100),M
100 CLS
110 PUT (219*RND,99*RND),M
120 FOR I=1 TO 1000:NEXT:GOTO 100
```

PROGRAMME 9.2

Avec le programme 9.3 nous allons étudier comment on peut utiliser GET et PUT pour recopier un motif plusieurs fois. Les lignes 30 à 80 dessinent six lignes qui simulent une partie d'un mur en briques.

```

0  *** 9-3 : Draws a wall **
10 DIM M(100)
20 SCREEN 1:COLOR 0,0:CLS
25 LINE (0,0)-(20,15),2,BF
30 LINE(20,0)-(20,5)
40 LINE (20,10)-(20,15)
50 LINE (0,10)-(20,10)
60 LINE (0,5)-(20,5)
70 LINE (10,0)-(10,5)
80 LINE (6,5)-(6,10)
90 LINE(16,5)-(16,10)
100 GET (0,1)-(20,10),M
110 FOR I=30 TO 200 STEP 21
120   FOR J=30 TO 190 STEP 10
130     PUT (I,J),M
140   NEXT
150 NEXT
160 LINE(30,30)-(218,199),3,B

```

PROGRAMME 9.3

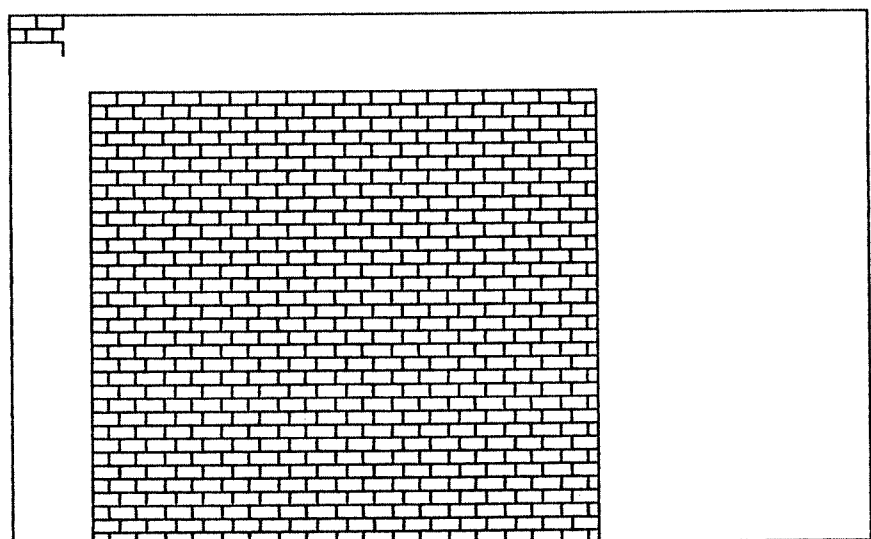


Fig. 9.1

La ligne 100 met la figure (GET) dans le tableau M et les deux boucles FOR écrivent (PUT) la figure sur l'écran, construisant un mur. Si huit instructions étaient nécessaires pour dessiner la figure d'origine, imaginez combien de temps il aurait fallu pour dessiner le mur en entier sans GET et PUT !

9.1.1 L'instruction GET

La syntaxe de l'instruction GET est : `GET(colonne1,ligne1)-(colonne2,ligne2), nom-tableau`. *Colonne1,ligne1* et/ou *colonne2,ligne2* peuvent être les coordonnées sous forme absolue ou relative. *Nom-tableau* doit être un tableau numérique de n'importe quel type. Pour calculer la taille minimum du tableau nécessaire pour mémoriser la figure, il est nécessaire d'effectuer le calcul suivant :

$$\text{Nombre d'octets} = 4 + \text{INT}((X * \text{bitsparpixel} + 7) / 8) * Y$$

où

$$X = \text{ABS}(\text{colonne2} - \text{colonne1}) + 1,$$

$$Y = \text{ABS}(\text{ligne2} - \text{ligne1}) + 1,$$

et *bitsparpixel* est 2 en moyenne résolution. Le nombre d'octets par élément d'un tableau est le suivant :

ENTIER 2 octets

SIMPLE PRECISION 4 octets

DOUBLE PRECISION 8 octets

Rappelez-vous également que, à moins que ce ne soit spécifié par l'instruction OPTION BASE 1, le 0ème élément de chaque dimension est disponible dans tout tableau². Par exemple, en moyenne résolution, pour exécuter l'instruction `GET(100,80)-(115,92),T`, le nombre d'octets nécessaire est :

$$X = 115 - 100 + 1 = 16$$

$$Y = 92 - 80 + 1 = 13$$

$$\text{Nombre d'octets} = 4 + \text{INT}((16 * 2 + 7)/8) * 13 = 56.$$

Si le type d'un tableau est entier, la DIMension minimum nécessaire est 27, si le type est simple précision, la DIM minimum est 13 et si le type

est double précision la DIM minimum est 6. Les tableaux peuvent être plus grands que le minimum exigé, mais s'ils sont plus petits ils provoqueront une erreur d'appel de fonction non autorisée.

Une fois que le tableau a été utilisé dans une instruction GET, si on change n'importe quel élément dans n'importe laquelle des variables à l'intérieur de la taille minimum exigée, on modifiera la figure. Par exemple au programme 9.3 si la ligne :

```
105 M(1)=43
```

est ajoutée, l'instruction PUT provoquera une erreur d'appel de fonction non autorisée.

9.1.2 L'instruction PUT

Une fois qu'une image a été mémorisée dans un tableau avec GET, elle peut être recopiée sur l'écran avec PUT. La syntaxe de cette instruction est `PUT(colonne, ligne), nom-tableau, mode`. *Colonne* et *ligne* doivent faire partie des intervalles ((0,0) à (319,199) ou (0,0) à (639, 199)) et l'image mémorisée dans le tableau doit tenir entièrement sur l'écran. Après avoir exécuté l'instruction `GET(0,0)–(9,10),M`, par exemple, l'image peut être réécrite (PUT) sur l'écran à n'importe quel point (X,Y) tel que $0 \leq X + 9 \leq 319$ (ou 639 en haute résolution), et $0 \leq Y + 10 \leq 199$.

Outre la possibilité de repositionner des figures sur l'écran, PUT peut reconstituer la figure suivant l'un des cinq modes suivants :

PSET. L'image est copiée exactement comme elle a été copiée par GET, toute information graphique précédente dans la zone affectée est perdue.

PRESET. L'image est reproduite comme le négatif de l'original. C'est-à-dire que les codes couleurs sont complémentés, changés en ceux que l'on voit à la table 9.1. Comme avec le mode PSET toute information graphique précédente dans la zone affectée est perdue.

Table 9.1. Couleurs complémentés dans PRESET

COULEUR ORIGINALE	COULEUR COMPLEMENT
3	0
2	1
1	2
0	3

XOR exécute un OU exclusif (voir appendice B) entre l'image et l'information graphique sur l'écran. Les couleurs résultantes sont affectées par le contenu précédent de la zone affectée comme le montre la table 9.2.

Table 9.2. Couleurs XOR

		<i>Valeur dans la figure</i>			
		0	1	2	3
Valeur	0	0	1	2	3
	1	1	0	3	2
sur	2	2	3	0	1
l'écran	3	3	2	1	0

Quand on exécute un OU exclusif sur une figure deux fois sur la même position, le contenu d'origine de l'écran est rétabli, ce qui est le résultat de peindre et effacer la figure. Ceci sera particulièrement utile pour l'animation, comme on le verra au paragraphe 10.4.1.

AND exécute un ET (voir appendice B) entre l'image et l'information graphique de l'écran. Le résultat est que seules les couleurs du premier plan de la figure seront affichées dans les parties de l'écran dont les couleurs étaient différentes de 0. La table 9.3 montre les couleurs résultantes.

Table 9.3. Couleurs AND

		<i>Valeur dans la figure</i>			
		0	1	2	3
Valeur	0	0	0	0	0
	1	0	1	0	1
sur	2	0	0	2	2
l'écran	3	0	1	2	3

OR exécute un OU (voir appendice B) entre l'image et l'information graphique de l'écran. Le résultat est que l'image existante et la nouvelle sont mélangées. La table 9.4 montre les couleurs résultantes.

Table 9.4. Couleurs OR

		<i>Valeur dans la figure</i>			
		0	1	2	3
Valeur	0	0	1	2	3
	1	1	1	3	3
sur	2	2	3	2	3
l'écran	3	3	3	3	3

9.2 LA COPIE DE LIGNES

Dans ce paragraphe nous verrons comment tracer une droite (ou une figure constituée de segments de droites) et comment l'effacer, restau-

rant la partie du fond qui a été effacée. Pour garder en mémoire la couleur de chaque point couvert par la droite, nous devons utiliser la méthode décrite au paragraphe 2.6.1 dans lequel la droite est tracée point par point, et non à l'aide de l'instruction LINE. Avant de tracer chaque point nous mémorisons sa couleur (que nous obtenons avec la fonction POINT) dans un tableau. Pour effacer la ligne et rétablir le fond, nous traçons à nouveau la droite point par point, mais cette fois en allant rechercher la couleur de chaque point dans le tableau que nous avons constitué.

Le programme 9.4 trace une droite entre les deux points dont les coordonnées sont rentrées aux lignes 20 et 30. Les lignes 510 à 530 du sous-programme 500 calculent les paramètres nécessaires pour tracer la droite. Selon le mode (PLOT ou RESTORE) une des deux sections suivantes est exécutée :

Les lignes 550 à 595 tracent la droite dans la couleur 3, la couleur de chaque point étant auparavant rangée dans le tableau COL.
Les lignes 600 à 640 retracent tous les points qui ont été effacés par la ligne, en utilisant les couleurs mémorisées dans le tableau COL.

```

0  ' ** 9-4 : Copy line **
10 SCREEN 1:COLOR 0,0:CLS:PLOT=1:
   RESTR=2:DIM COL(10,376)
20 INPUT"First point (X,Y) ";X1,Y1
30 INPUT"Second point (X,Y) ";X2,Y2
40 CLS
50 FOR I=1 TO 20
60   PRINT"This is just a test; nothing else."
70 NEXT
80 MODE=PLOT:GOSUB 500
90 W$=INPUT$(1)
100 MODE=RESTR:GOSUB 500
110 END
500 ' Plot and erase line
510 DX=X2-X1:DY=Y2-Y1
520 D=SQR(DX*DX+DY*DY)
530 SX=DX/D:SY=DY/D
540 ON MODE GOTO 550,630
550 'Plot line
560 FOR I=0 TO D
570   COL(K,I)=POINT(X1+SX*I,Y1+SY*I)
580 NEXT
590 FOR I=1 TO D
600   PSET(X1+SX*I,Y1+SY*I),2
610 NEXT
620 RETURN
630 'Restore line
640 FOR I=0 TO D
650   PSET(X1+SX*I,Y1+SY*I),COL(K,I)
660 NEXT
670 RETURN

```

PROGRAMME 9.4

La droite la plus longue ³ a une longueur de 376 (de (0,0) à (319,199)), donc l'instruction DIM de la ligne réserve suffisamment de place pour

n'importe quel cas. Une seule ligne est tracée dans ce programme, mais l'emplacement mémoire pour 11 lignes (de 0 à 10) est réservé dans le tableau COL. Les sous-programmes de traçage et de restitution du programme 9.4 se limitent tous deux à $K=0$, mais on utilise les autres dimensions du tableau dans les programmes suivants.

Notez que les couleurs de tous les points de la droite ont été entrées dans le tableau avant de tracer les points. Bien qu'on aurait pu traiter les deux opérations plus rapidement avec une simple boucle FOR, dans ce cas particulier elles doivent être exécutées séparément : du fait des imprécisions d'arrondis, deux points peuvent avoir les mêmes coordonnées. Le premier point sera copié et tracé correctement mais quand le second point sera tracé, la couleur retournée par POINT correspondra à celle du point tracé par le PSET précédent, et quand la ligne sera restaurée certains points ne correspondront pas à ceux d'origine. Un tel cas est illustré par la droite (0,0) à (100,100) ; essayez-le en supprimant les lignes 580 et 590 du programme 9.4.

9.3 L'EDITEUR DE LIGNES

Une manipulation fastidieuse dans la création graphique est le positionnement correct des lignes sur des fonds non vides. Pour chaque ligne, il faut rentrer les points de début et de fin avant de voir apparaître le graphique résultant. La plupart du temps les premières tentatives ne marcheront pas, et empiriquement la ligne sera emmenée de plus en plus près de sa position finale correcte. Quand il y a plusieurs lignes à traiter, le processus devient très long.

En utilisant le sous-programme 500 du programme 9.4 nous développerons un éditeur pour la création et la modification interactives de lignes, avec la possibilité de restitution du fond.

Le programme 9.5 commence par le dessin de la droite (160,100) à (170,110). On entre un caractère en ligne 30. Les chiffres correspondant aux quatre flèches du pavé numérique provoquent le déplacement d'un des deux points limites en ajoutant le contenu de INC (qui débute avec la valeur 10) aux coordonnées de l'un des points. On utilise la variable PT pour déterminer lequel des deux points est affecté : quand $PT = 1$, c'est le premier ($X1, Y1$), quand $PT = 2$, c'est le second. Le programme commence avec $PT=1$. En appuyant sur la touche « 9 », on change PT en 2 et si on appuie sur la touche « 7 », on remet PT à 1. Pour changer la valeur de INC, appuyez sur le signe « + » (increment) ou sur le signe « - » (decrement).

En ligne 25, juste après qu'on ait affecté aux variables $X1, Y1, X2$ et $Y2$ leurs valeurs initiales, la droite joignant ces points est tracée en appelant le sous-programme 410. Celui-ci, non seulement trace la droite, mais aussi mémorise dans COL les couleurs des points affectés par la droite. Chaque fois qu'un des points limites est changé, la ligne est d'abord

restaurée (les couleurs du fond sont restituées) et ensuite tracées pour refléter le changement qui vient d'être opéré. Quand on ne veut plus rien modifier, <return> (CHR\$(13)) laisse la ligne telle quelle, et réaffecte les valeurs aux points limites pour recommencer avec une seconde droite.

```

0 ' ** 9-5 : Line editor **
10 SCREEN 1:COLOR 0,0:CLS:PLOT=1:
   RESTR=2:DIM COL(10,376)
20 X1=160:Y1=100:X2=170:Y2=110
25 INC=10:PT=1:GOSUB 410
30 W$=INPUT$(1):
   IF W$=CHR$(13)
   THEN
     20
31 IF W$="+"
   THEN
     IF INC<20
     THEN
       INC=INC+1:GOTO 30
     ELSE BEEP:GOTO 30
32 IF W$="-"
   THEN
     IF INC>1
     THEN
       INC=INC-1:GOTO 30
     ELSE BEEP:GOTO 30
33 IF W$<>"B"AND W$<>"4"AND W$<>"6"AND W$<>"2"
   THEN SOUND 2000,1:GOTO 30
35 ON VAL(W$)+1 GOTO 30,30,40,30,50,30,60,70,80,90
36 BEEP:GOTO 30
40 ON PT GOTO 42,44
42 GOSUB 400:Y1=Y1+INC:GOSUB 410:GOTO 30
44 GOSUB 400:Y2=Y2+INC:GOSUB 410:GOTO 30
50 ON PT GOTO 52,54
52 GOSUB 400:X1=X1-INC:GOSUB 410:GOTO 30
54 GOSUB 400:X2=X2-INC:GOSUB 410:GOTO 30
60 ON PT GOTO 62,64
62 GOSUB 400:X1=X1+INC:GOSUB 410:GOTO 30
64 GOSUB 400:X2=X2+INC:GOSUB 410:GOTO 30
70 PT=1:GOTO 30
72 GOTO 30
80 ON PT GOTO 82,84
82 GOSUB 400:Y1=Y1-INC:GOSUB 410:GOTO 30
84 GOSUB 400:Y2=Y2-INC:GOSUB 410:GOTO 30
90 PT=2:GOTO 30
400 MODE=RESTR:GOSUB 500:RETURN
410 MODE=PLOT:GOSUB 500:RETURN
500 Plot and erase line
510 DX=X2-X1:DY=Y2-Y1
520 D=SQR(DX*DX+DY*DY)
530 SX=DX/D:SY=DY/D
540 ON MODE GOTO 550,700
550 'Plot line
560 FOR I=0 TO D
570 COL(K,I)=POINT(X1+SX*I,Y1+SY*I)
580 NEXT
590 FOR I=0 TO D
600 PSET(X1+SX*I,Y1+SY*I),2
610 NEXT

```



```

620 RETURN
700 'Restore line
710 FOR I=0 TO D
720   PSET(X1+SX*I,Y1+SY*I),COL(K,I)
730 NEXT
740 RETURN

```

PROGRAMME 9.5

Bien que des figures plutôt complexes puissent être créées avec le programme 9.5 et un peu de pratique, il est très gênant de ne pas savoir quel point (le premier ou le second) sera affecté par le prochain mouvement du curseur et de combien il sera déplacé. Il serait très pratique d'avoir une petite fenêtre où cette information pourrait être lue chaque fois qu'on en a besoin. Toutefois cela réduirait significativement la taille de l'écran.

Une manière facile (et naïve) de résoudre ce problème est de copier dans un tableau à deux dimensions la couleur de chaque point de la partie qui va être utilisée comme fenêtre, imprimer les valeurs désirées, et quand on n'a plus besoin de ces valeurs, rétablir le contenu original de la fenêtre. Le minimum d'information dont on aura besoin sera le point actif en cours, (qui peut être signalé par un 1 ou un 2) et un nombre à deux chiffres avec l'incrément en cours (que nous avons décidé de restreindre à la fourchette de 1 à 20). Ces deux valeurs seules nécessitent deux tableaux, un de 8 par 8 et un autre de 8 par 16. Outre le gaspillage de mémoire (en utilisant des tableaux d'entiers, cette fenêtre à trois caractères exigera $24 \times 8 \times 2 = 384$ octets), le processus sera très lent. Au paragraphe 9.6 nous étudierons une manière pour gérer les fenêtres plus efficacement, avec GET et PUT, comme vous devez vous en douter.

9.4 LE DEPLACEMENT DE POLYGONES

Dans ce paragraphe nous allons étendre la méthode de copie et restitution de lignes expliquée au paragraphe précédent à des figures plus complexes.

Le programme 9.6 trace un pentagone dont les coordonnées ont été calculées en utilisant la méthode expliquée au paragraphe 3.4.3 et entrées ici en DATA, afin de simplifier le programme. Les sous-programmes 400 et 410 (qui à leur tour appellent le sous-programme 500) sont les sous-programmes de traçage et de restauration du programme 9.5.

```

0  "" 9-6 : Moves polygons ""
10 SCREEN 1:COLOR 0,0:CLS:PLOT=1:RESTR=2:DIM COL(10,376)
13 FOR I=1 TO 20
15   PRINT"This is another test."
18 NEXT

```

```

20 READ X1:READ Y1
30 FOR K=0 TO 4
40   READ X2,Y2
50   GOSUB 410
60   X1=X2:Y1=Y2
70 NEXT
80 W$=INPUT$(1)
90 RESTORE:READ X1,Y1
100 FOR K=0 TO 4
110   READ X2,Y2
120   GOSUB 400
130   X1=X2:Y1=Y2
140 NEXT
150 END
200 DATA 250,100,188,186,87,153,87,47,188,14,250,100

```

PROGRAMME 9.6

9.5 LA COPIE DE CERCLES

Le processus pour sauvegarder des cercles est semblable à celui utilisé pour mémoriser les lignes, la seule différence étant dans le beaucoup plus grand nombre de points à mémoriser. En coordonnées polaires, ainsi qu'il est expliqué au paragraphe 3.4.3, un cercle peut être modélisé par des petits segments de droites. En combinant cette méthode et la mémorisation des lignes décrite au paragraphe 9.2, nous sommes maintenant capables de dessiner un cercle avec la possibilité de restituer le fond original.

Pour économiser du temps et de la place, au lieu de tracer tout le cercle, nous allons utiliser un grand pas de telle sorte que seuls quelques points seront tracés. Ceci est suffisant pour donner une idée du cercle final et économise de la mémoire et du temps d'exécution. Une fois que le cercle est jugé satisfaisant, dans le programme 9.7, le cercle définitif peut être tracé en appuyant sur <return>.

```

0  ** 9-7 : Circle editor **
10 SCREEN 1:COLOR 0,0:CLS
15 DEFINT A,C
20 F=1.745329E-02:DIM COL(50)
40 XC=160:YC=100:R=50
50 GOSUB 500
60 W$=INPUT$(1)
70 IF W$="+"
   THEN
     GOSUB 700:R=R+10:GOSUB 500:GOTO 60
80 IF W$="-"
   THEN
     GOSUB 700:R=R-10:GOSUB 500:GOTO 60

```

```

90 IF W$="8"
    THEN
        GOSUB 700:YC=YC-10:GOSUB 500:GOTO 60
100 IF W$="2"
    THEN
        GOSUB 700:YC=YC+10:GOSUB 500:GOTO 60
110 IF W$="4"
    THEN
        GOSUB 700:XC=XC-10:GOSUB 500:GOTO 60
120 IF W$="6"
    THEN
        GOSUB 700:XC=XC+10:GOSUB 500:GOTO 60
130 IF W$(<>CHR$(13))
    THEN
        190
140 FOR A=0 TO 360 STEP 2
150   X=R*COS(A*F)
160   Y=R*SIN(A*F)
170   IF A=0
        THEN
            PSET(X+XC,Y+YC)
        ELSE
            LINE-(X+XC,Y+YC)
180 NEXT:GOTO 40
190 BEEP:GOTO 60
500 '** plot circle **
510 C=0
520 FOR A=0 TO 350 STEP 10
530   X=R*COS(A*F)
540   Y=R*SIN(A*F)
545   TEMCOL=POINT(X+XC,Y+YC):
        IF TEMCOL=-1
            THEN
                TEMCOL=0
550   COL(C)=TEMCOL
560   PSET(X+XC,Y+YC),2
570   C=C+1
580 NEXT
590 RETURN
700 '** Restore circle **
710 C=0
720 FOR A=0 TO 350 STEP 10
730   X=R*COS(A*F)
740   Y=R*SIN(A*F)
750   PSET(X+XC,Y+YC),COL(C)
760   C=C+1
770 NEXT
780 RETURN

```

PROGRAMME 9.7

En se servant de la technique expliquée au paragraphe 3.4.2, on peut étendre cette méthode pour réaliser des ellipses.

9.6 LES FENETRES

Il y a beaucoup de situations où l'on désire isoler une partie d'une image pour faciliter son étude et la manipuler. En utilisant une combinaison de PUT et GET, des parties de l'écran peuvent être sauvegardées temporairement tandis que le reste de l'écran est affecté par des nouvelles instructions.

Supposez, par exemple, qu'une partie de la figure 9.1 soit agrandie en utilisant la méthode expliquée au paragraphe 7.11. Le programme 9.8 utilise les chiffres des quatre touches de déplacement du curseur⁴ pour isoler une partie de l'écran en déplaçant les quatre bords, le signe « + » pour régler l'incrément du bord à 10, et le signe « - » pour le régler à 1. Quand la taille du cadre est considérée comme correcte (indiqué par <return>), la fenêtre est copiée dans le tableau M, sans le cadre. L'écran est effacé et le tableau est réécrit (PUT) sur l'écran. Le processus d'agrandissement peut alors commencer avec un écran ne comportant plus que la zone sélectionnée.

Ce programme n'établit pas le mode graphique et n'efface pas l'écran dans le but de traiter, en le laissant inchangé, ce qui a été produit par un autre programme, qui à son tour doit appeler le programme 9.8 avec l'instruction RUN « 9-8 » ou quelque autre nom que vous lui avez donné. Une alternative serait l'instruction CHAIN, s'il y avait des variables communes aux deux programmes. La meilleure manière de tester le programme est la suite d'instructions :

```
SCREEN 1
LIST
RUN
```

```
0  ' ** 9-8 : Copy window **
10 X1=-1:Y1=-1:X2=320:Y2=200
15 DIM M(4000):D=1:COL=3
20 W$=INPUT$(1)
30 IF W$="+"
    THEN
        D=10:GOTO 20
40 IF W$="-"
    THEN
        D=1:GOTO 20
50 IF W$="8"
    THEN
        Y1=Y1+D:LINE(X1,Y1)-(X2,Y1),COL:GOTO 20
60 IF W$="2"
    THEN
        Y2=Y2-D:LINE(X1,Y2)-(X2,Y2),COL:GOTO 20
70 IF W$="4"
    THEN
        X1=X1+D:LINE(X1,Y1)-(X1,Y2),COL:GOTO 20
```

```

80 IF W$="6"
    THEN
        X2=X2-D:LINE(X2,Y1)-(X2,Y2),COL:GOTO 20
90 IF W$<>"G"AND W$<>"c"
    THEN
        100
93 COL=COL+1:
    IF COL=4
        THEN
            COL=1
95 LINE(X1,Y1)-(X2,Y2),COL,B:GOTO 20
100 IF W$=CHR$(13)
    THEN
        GET(X1+1,Y1+1)-(X2-1,Y2-1),M:GOTO 200
110 SOUND 1000,.1:GOTO 20
200 CLS:PUT(X1+1,Y1+1),M

```

PROGRAMME 9.8

Une autre application utile des instructions GET et PUT est la création de fenêtres de texte. Quand tout l'écran est occupé, les messages ou les instructions ne peuvent pas être facilement affichées sans affecter ce qui l'est déjà. Si on n'a besoin que de peu de messages, ils peuvent être transmis par des signaux sonores clairement distincts, comme ceux produits par le programme 9.9, une version très simple d'une table traçante. La table 9.5 montre les commandes de déplacement du curseur.

Table 9.5

<i>Touche</i>	<i>Effet</i>
8	Vers le haut
2	Vers le bas
4	Vers la gauche
6	Vers la droite
+	Régler l'incrément à 10
-	Régler l'incrément à 1
C	Changer la couleur

```

0  ** 9-9 : Sketching board **
5  SCREEN 1:COLOR 1,0:CLS
10 X=160:Y=100:COL=2:D=2:GOTO 500
20 W$=INPUT$(1)
30 IF W$="+"
    THEN
        D=10:GOTO 20
40 IF W$="-"
    THEN
        D=2:GOTO 20
50 IF W$="B"
    THEN
        IF Y-D<0

```

```

        THEN
            SOUND 5000,1:GOTO 20
        ELSE
            Y=Y-D:GOTO 500
60 IF W$="2"
    THEN
        IF Y+D>199
            THEN
                SOUND 5000,1:GOTO 20
            ELSE
                Y=Y+D:GOTO 500
70 IF W$="4"
    THEN
        IF X-D<0
            THEN
                SOUND 5000,1:GOTO 20
            ELSE
                X=X-D:GOTO 500
80 IF W$="6"
    THEN
        IF X+D>319
            THEN
                SOUND 5000,1:GOTO 20
            ELSE
                X=X+D:GOTO 500
90 IF W$<>"C"AND W$<>"c"
    THEN
        110
93 COL=COL+1:
    IF COL=4
        THEN
            COL=0
95 GOTO 500
110 SOUND 100,2:GOTO 20
200 CLS:PUT(X1+1,Y1+1),M,PRESET
500 LINE(X,Y)-(X+1,Y+1),COL,BF:GOTO 20

```

PROGRAMME 9.9

Un son aigu indique une tentative de tracer un point en dehors de l'écran, alors qu'un son grave signale une commande inexistante.

Quand il y a une grande variété de messages ou que les messages eux-mêmes sont plus complexes que de simples codes oui-ou-non, il faut trouver une manière de les transmettre sans occuper l'écran en permanence. Une solution possible est l'imprimante. Si chaque message y est envoyé, il est possible d'avoir autant de messages — aussi complexes soient-ils — que l'on veut, sans toucher à l'écran. Toutefois tout le monde n'a pas une imprimante, le papier est gaspillé et se rendre à l'endroit où se trouve l'imprimante et peut être déplacer le papier pour voir le dernier message n'est pas très pratique.

Une autre façon de régler le problème est de mémoriser temporairement une zone de l'écran. y imprimer les messages dont on a besoin, et rétablir l'affichage d'origine une fois qu'on n'a plus besoin du texte. Le programme 9.10 utilise cette approche dans une version revue du programme 9.9, cette fois avec la possibilité de changer la taille du curseur

(le signe plus agrandit le curseur et le signe moins le réduit). Le point d'interrogation (« ? ») est une fonction qui aide un peu en affichant les commandes disponibles pour le mouvement du curseur et le changement de couleur, de même que la taille courante du curseur. La fenêtre dans laquelle ce texte doit être imprimé est mémorisée dans le tableau M, et effacée avec l'instruction LINE(0,0)-(80,80),0,BF. Notez que l'instruction LINE utilisée avec la couleur du fond dans le mode BF est équivalente à CLS, à la différence que la zone effacée peut être plus petite que la totalité de l'écran. L'instruction LINE(0,0)-(80,80),3,B trace un cadre autour de la fenêtre pour la différencier du reste de l'image. Quand on appuie sur une touche, la fenêtre est effacée et le contenu original de l'écran est rétabli en réécrivant (PUT)M sur l'écran. L'instruction PUT doit être utilisée avec le mode PSET ou la fenêtre restaurée ne correspondra pas à l'original.

```

0  '** 9-10 : Board with help **
5  SCREEN 1:COLOR 1,0:CLS
7  DIM M(2000)
10 X=160:Y=100:COL=2:D=1:GOTO 500
20 W$=INPUT$(1)
22 IF W$<>"?"
    THEN
        30
    ELSE
        GET(0,0)-(80,80),M
23  LINE(0,0)-(80,80),0,BF:LINE(0,0)-(80,80),3,B
24  LOCATE 2,2:PRINT"8-Up":LOCATE 3,2:
    PRINT"2-Down":LOCATE 4,2:PRINT"4-Left":
    LOCATE 5,2:PRINT"6-Right"
25  LOCATE 6,2:PRINT"+ Grow":LOCATE 7,2:
    PRINT"- Shrink":LOCATE 8,2:PRINT"C-Color"
26  LOCATE 9,2:PRINT"Size:";D
27  W$=INPUT$(1)
28  PUT(0,0),M,PSET:GOTO 22
30  IF W$="+"
    THEN
        D=2*D:
        IF D=32
            THEN
                D=1:GOTO 500
            ELSE
                500
40  IF W$="-"
    THEN
        D=D*.5:
        IF D<1
            THEN
                D=8:GOTO 500
            ELSE
                500
50  IF W$="8"
    THEN
        IF Y-D<0
            THEN
                SOUND 5000,1:GOTO 20
            ELSE

```

```

        Y=Y-D:GOTO 500
60 IF W$="2"
    THEN
        IF Y+D>199
            THEN
                SOUND 5000,1:GOTO 20
            ELSE
                Y=Y+D:GOTO 500
70 IF W$="4"
    THEN
        IF X-D<0
            THEN
                SOUND 5000,1:GOTO 20
            ELSE
                X=X-D:GOTO 500
80 IF W$="6"
    THEN
        IF X+D>319
            THEN
                SOUND 5000,1:GOTO 20
            ELSE
                X=X+D:GOTO 500
90 IF W$<>"C"AND W$<>"c"
    THEN
        110
93 COL=COL+1:
    IF COL=4
        THEN
            COL=0
95 GOTO 500
110 SOUND 100,2:GOTO 20
200 CLS:PUT(X1+1,Y1+1),M,PRESET
500 LINE(X,Y)-(X+D-1,Y+D-1),COL,BF:GOTO

```

PROGRAMME 9.10

Nous pouvons maintenant réécrire l'éditeur de lignes du paragraphe 9.3 avec une nouvelle approche. Si l'écran en entier est vu comme une fenêtre, on peut le mémoriser dans n'importe quel tableau tandis qu'on utilise l'éditeur. Quand on considère que la position d'une ligne est fixée, <return> provoque une recopie d'écran, contenant cette fois la nouvelle ligne. Notez que dans le programme 9.11 les lignes sont tracées avec l'instruction LINE (beaucoup plus rapidement qu'avec la méthode point par point) et aussi que nous avons rajouté une commande (« C ») pour changer la couleur.

```

0  ** 9-11 : Line editor with GET and PUT **
10 SCREEN 1:COLOR 0,0:CLS:PLOT=1:RESTR=2:
    DIM COL(10,376),M(4000)
20 X1=160:Y1=100:X2=170:Y2=110:COL=2
25 INC=10:PT=1:GOSUB 420:GOSUB 410
30 W$=INPUT$(1):
    IF W$=CHR$(13)
        THEN
            20
31 IF W$="+"
    THEN

```



```

        IF INC<20
        THEN
            INC=INC+1:GOTO 30
        ELSE
            BEEP:GOTO 30
32 IF W$="-"
    THEN
        IF INC>1
        THEN
            INC=INC-1:GOTO 30
        ELSE
            BEEP:GOTO 30
33 IF W$<>"C"AND W$<>"c"
    THEN
        35
    ELSE
        COL=COL+1:
        IF COL=4
        THEN
            COL=0
34 GOSUB 410:GOTO 30
35 ON VAL(W$)+1 GOTO 36,30,40,30,50,30,60,70,80,90
36 BEEP:GOTO 30
40 ON PT GOTO 42,44
42 GOSUB 400:Y1=Y1+INC:GOSUB 410:GOTO 30
44 GOSUB 400:Y2=Y2+INC:GOSUB 410:GOTO 30
50 ON PT GOTO 52,54
52 GOSUB 400:X1=X1-INC:GOSUB 410:GOTO 30
54 GOSUB 400:X2=X2-INC:GOSUB 410:GOTO 30
60 ON PT GOTO 62,64
62 GOSUB 400:X1=X1+INC:GOSUB 410:GOTO 30
64 GOSUB 400:X2=X2+INC:GOSUB 410:GOTO 30
70 PT=1:GOTO 30
72 GOTO 30
80 ON PT GOTO 82,84
82 GOSUB 400:Y1=Y1-INC:GOSUB 410:GOTO 30
84 GOSUB 400:Y2=Y2-INC:GOSUB 410:GOTO 30
90 PT=2:GOTO 30
400 PUT(0,0),M,PSET:RETURN
410 LINE(X1,Y1)-(X2,Y2),COL:RETURN
420 GET(0,0)-(319,199),M:RETURN
500 ' Plot and erase line
510 DX=X2-X1:DY=Y2-Y1
520 D=SQR(DX*DX+DY*DY)
530 SX=DX/D:SY=DY/D
540 ON MODE GOTO 550,700
550 'Plot line
560 FOR I=0 TO D
570   COL(K,I)=POINT(X1+SX*I,Y1+SY*I)
580 NEXT
590 FOR I=0 TO D
600   PSET(X1+SX*I,Y1+SY*I),2
610 NEXT
620 RETURN
700 'Restore line
710 FOR I=0 TO D
720   PSET(X1+SX*I,Y1+SY*I),COL(K,I)
730 NEXT
740 RETURN

```

PROGRAMME 9.11

9.7 LE NEGATIF

Quand une partie de l'écran qui a été mémorisée dans un tableau est réécrite (PUT) sous le mode PRESET, toutes les couleurs sont complémentées et le résultat est un négatif de l'original. On peut voir les valeurs des couleurs résultantes à la table 9.1 et la table 9.6 montre les noms des couleurs dans le réglage particulier

COLOR 1,0

Table 9.6

COULEUR D'ORIGINE	COMPLEMENT
BLEU	JAUNE
VERT	ROUGE
ROUGE	VERT
JAUNE	BLEU

Pour obtenir le négatif de la totalité de l'écran, on doit d'abord le copier avec l'instruction GET(0,0)-(319,199),Nom-de-tableau,PRESET. En haute résolution les couleurs sont simplement inversées, c'est-à-dire que le blanc devient noir et vice versa. Pour avoir tout l'écran sous ce mode, l'instruction est GET(0,0)-(639,199),Nom-de-tableau. La figure 9.2 nous montre le mur de brique, qu'on avait vu à l'origine à la figure 9.1, après qu'il ait été copié (GET) et réécrit (PUT) sous le mode PRESET.

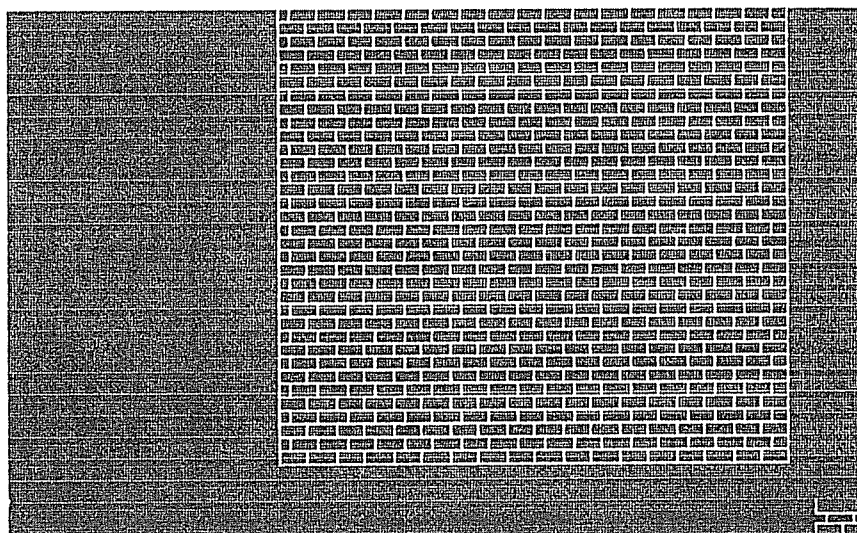


Fig. 9.2

Le programme 9.12 utilise la même technique que le programme 9.8 pour isoler une partie de l'écran. Quand on appuie sur <return>, le négatif de la fenêtre choisie apparaît et le reste de l'écran est effacé. Rappelez-vous que ce programme ne travaille pas en mode graphique et n'efface pas l'écran. Si l'écran est en mode texte quand le programme est exécuté, une erreur d'appel de fonction non autorisée fera échouer le programme.

```

0  '** 9-12 : Negative section **
10 X1=-1:Y1=-1:X2=320:Y2=200
15 DIM M(4000):D=1:COL=3
20 W$=INPUT$(1)
30 IF W$="+"
    THEN
        D=10:GOTO 20
40 IF W$="-"
    THEN
        D=1:GOTO 20
50 IF W$="8"
    THEN
        Y1=Y1+D:LINE(X1,Y1)-(X2,Y1),COL:GOTO 20
60 IF W$="2"
    THEN
        Y2=Y2-D:LINE(X1,Y2)-(X2,Y2),COL:GOTO 20
70 IF W$="4"
    THEN
        X1=X1+D:LINE(X1,Y1)-(X1,Y2),COL:GOTO 20
80 IF W$="6"
    THEN
        X2=X2-D:LINE(X2,Y1)-(X2,Y2),COL:GOTO 20
90 IF W$<>"C"AND W$<>"c"
    THEN
        100
93 COL=COL+1:
    IF COL=4
    THEN
        COL=1
95 LINE(X1,Y1)-(X2,Y2),COL,B:GOTO 20
100 IF W$=CHR$(13)
    THEN
        GET(X1+1,Y1+1)-(X2-1,Y2-1),M:GOTO 200
110 SOUND 1000,.1:GOTO 20
200 CLS:PUT(X1+1,Y1+1),M,PRESET

```

PROGRAMME 9.12

9.8 LE DEROULEMENT DE L'ECRAN

Comme on l'a expliqué au paragraphe 7.4, quand un caractère est imprimé sur la position la plus à droite de la 24ème ou 25ème ligne, l'écran est déplacé vers le haut, c'est-à-dire que toutes les lignes sont déplacées vers le haut (le contenu de la ligne *i* est placé à la ligne *i*-1, sauf pour la ligne 25 qui n'est jamais déplacée), et la ligne 24 est effacée. Nous allons déplacer un écran vers le haut à partir de l'intérieur d'un programme sans écrire quoi que ce soit sur la position la plus à droite

de la ligne 24 ou 25. La méthode pour obtenir ce résultat ⁵ est directe : commencer par copier la seconde ligne en entier dans un tableau, et l'écrire sur la ligne 1 sous le mode PSET (tout autre mode ne redonnera pas une reproduction fidèle de l'original), déplacer la ligne 3 à la ligne 2, et ainsi de suite. A la fin du processus, la ligne 25 peut être effacée par l'instruction LINE(0,192)-(319,199),0,BF en moyenne résolution ou LINE(0,192)-(639,199),0,BF en haute résolution. Le programme 9.13 utilise la méthode que l'on vient de décrire pour déplacer l'écran vers le haut.

```
0 *** 9-13 : Scroll up **
10 DEFINT A-Z
20 DIM M(500)
30 FOR I=1 TO 8
40   FOR J=8 TO 199 STEP 8
50     GET(0,J)-(319,J+7),M
60     PUT(0,J-8),M,PSET
70   NEXT
80   LINE(0,192)-(319,199),0,BF
90 NEXT
```

PROGRAMME 9.13

Il est aussi possible de déplacer l'écran vers le bas en inversant simplement les paramètres de la boucle FOR et en effaçant la première ligne, comme le montre le programme 9.14.

```
0 *** 9-14 : Scroll down **
10 DEFINT A-Z
20 DIM M(500)
30 FOR I=1 TO 24
40   FOR J=184 TO 0 STEP -8
50     GET(0,J)-(319,J+7),M
60     PUT(0,J+8),M,PSET
70   NEXT
80   LINE(0,0)-(319,7),0,BF
90 NEXT
```

PROGRAMME 9.14

Une manière inhabituelle de déplacer l'écran est sur les côtés : les programmes 9.15 et 9.16 déplacent l'écran sur la gauche et sur la droite.

```
0 *** 9-15 : Scroll left **
10 DEFINT A-Z
```

```

20 DIM M(500)
30 FOR I=1 TO 24
40   FOR J=8 TO 319 STEP 8
50     GET(J,0)-(J+7,199),M
60     PUT(J-8,0),M,PSET
70   NEXT
80   LINE(311,0)-(319,199),0,BF
90 NEXT

```

PROGRAMME 9.15

```

0  °° 9-16 : Scroll right °°
10 DEFINT A-Z
20 DIM M(500)
30 FOR I=1 TO 24
40   FOR J=311 TO 7 STEP -8
50     GET(J-7,0)-(J,199),M
60     PUT(J+1,0),M,PSET
70   NEXT
80   LINE(0,0)-(7,199),0,BF
90 NEXT

```

PROGRAMME 9.16

9.9 LE DEROULEMENT LENT

Une manière élégante de déplacer l'écran n'est pas par sauts de caractères, mais par sauts de pixels. De cette manière les lignes se déplacent lentement et c'est très facile à lire, même quand le texte se déplace. Le programme 9.17 déplace l'écran lentement vers le haut.

```

0  °° 9-17 : Smooth scroll °°
10 DEFINT A-Z
20 DIM M(500)
30 FOR I=1 TO 24
40   FOR J=1 TO 199
50     GET(0,J)-(319,J),M
60     PUT(0,J-1),M,PSET
70   NEXT
80   LINE(0,199)-(319,199),0,BF
90 NEXT

```

PROGRAMME 9.17

On peut utiliser la même technique pour dérouler l'écran vers le bas et sur les côtés. la méthode est lente mais peut être efficace quand vous déplacez des petits textes ou des fenêtres graphiques, comme dans les jeux où un petit écran d'ordinateur ou de télévision est simulé. Le programme 9.18 raconte une histoire dans une petite fenêtre.

```

0  '** 9-18 : Scrolling window **
10 DEFINT A-Z: DIM M(500)
20 SCREEN 1:CLS
30 LINE(201,11)-(300,67),3,B
40 LINE(203,13)-(298,65),3,B
50 FOR I=1 TO 6
60   READ W$
70   LOCATE I+2,27:PRINT W$;
80 NEXT
90 FOR I=1 TO 16
100  FOR L=1 TO 4
110   FOR J=17 TO 63 STEP 2
120    GET(205,J)-(296,J+1),M
130    PUT(205,J-2),M,PSET
140   NEXT
150   LINE(205,62)-(296,64),0,BF
160  NEXT
162  READ W$:LOCATE 8,27:PRINT W$;
164 NEXT
175 DATA ,,,,
180 DATA This is the,story of a,
    very small>window. She
190 DATA told people,stories and,
    tales that,scrolled.,,,
    " THE END",,,,,,

```

PROGRAMME 9.18

9.10 BOUCLAGE ET ENROULEMENT

Une des limitations de l'écran est que, étant un objet statique, il est impossible de s'y déplacer comme sur un morceau de papier. Heureusement, l'utilisation de GET et PUT permet certaines manipulations qui ajoutent une grande souplesse aux programmes. Nous allons d'abord traiter l'écran comme s'il était une bande de papier fermée qui peut être déplacée horizontalement comme le montre la figure 9.3.

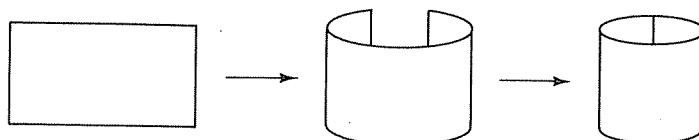


Fig. 9.3

Pour réaliser cela, nous pouvons utiliser la technique de déroulement présentée au paragraphe 9.8. Nous déplacerons l'écran sur les côtés, mais au lieu d'effacer la dernière colonne, nous la copierons sur la première de l'autre côté de l'écran. Voyons comment l'on s'y prend. Comme pour le déplacement de caractères, nous diviserons l'écran en sections ; toutefois maintenant nous ne sommes pas limités par la taille des caractères et nous pouvons donc choisir des sections plus grandes ou plus petites que la ligne de texte. Les flèches de la figure 9.4 montrent le mouvement des bandes en lesquelles on a divisé l'écran. Ici on l'a divisé en quatre sections, mais on peut prendre un autre nombre, de préférence multiple entier de 319 (ou 639).

Dans le cas extrême, seules deux sections peuvent être utilisées, le résultat étant dans un transfert rapide, mais nécessitant des tableaux de stockage importants. A l'autre extrême, le tableau peut être divisé en 320 colonnes. Le résultat sera une vitesse plus lente mais on pourra utiliser de tous petits tableaux. Bien sûr il y a beaucoup de situations entre les deux et il faut trouver un compromis entre la vitesse et la mémoire. Le programme 9.19 déplace l'écran sur la gauche et sur la droite avec un bouclage, le divisant en huit sections. Pour le déplacer vers la gauche, appuyez sur le « 1 » et pour le déplacer vers la droite sur le « 0 ».

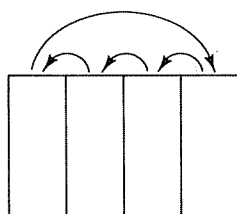


Fig. 9.4

```
0  ** 9-19 : Horizontal wrap around **
5  COL=1: DIM M(500), T(500)
10 SCREEN 1:CLS
20 FOR I=0 TO 199 STEP 5
30  LINE(0,0)-(159,I),COL
40  LINE(319,199)-(160,I),COL
50  COL=COL+1:
    IF COL=4
    THEN
        COL=1
60 NEXT
70 W$=INPUT$(1)
80 IF W$="1"
    THEN
        100
    ELSE
        IF W$="0"
        THEN
            200
```

```

ELSE
  70
100 GET(0,0)-(39,199),T
110 FOR I=40 TO 319 STEP 40
120   GET(I,0)-(I+39,199),M
130   PUT(I-40,0),M,PSET
140 NEXT
150 PUT(280,0),T,PSET:GOTO 70
200 GET(280,0)-(319,199),T
210 FOR I=240 TO 0 STEP -40
220   GET(I,0)-(I+39,199),M
230   PUT(I+40,0),M,PSET
240 NEXT
250 PUT(0,0),T,PSET:GOTO 70

```

PROGRAMME 9.19

Une seconde manière de déplacer l'écran est verticalement, ainsi il peut être considéré comme une bande fermée qui se déplace vers le haut ou le bas comme le montre la figure 9.5.

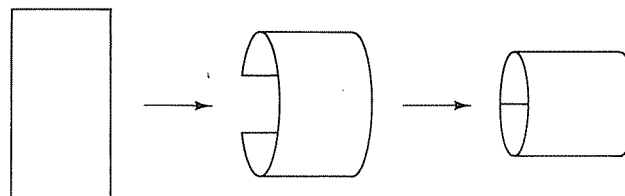


Fig. 9.5

Le programme 9.20 déplace l'écran vers le haut et le bas. Pour le déplacer vers le haut, appuyez sur le « 1 » et vers le bas sur le « 0 ».

```

0 ' ** 9-20 : Vertical wrap around **
5 COL=1:DIM M(900),T(900)
10 SCREEN 1:CLS
20 FOR I=0 TO 199 STEP 5
30   LINE(0,0)-(159,I),COL
40   LINE(319,199)-(160,I),COL
50   COL=COL+1:
    IF COL=4
      THEN
        COL=1
60 NEXT
70 W$=INPUT$(1)
80 IF W$="1"
    THEN
      300
    ELSE
      IF W$="0"
        THEN
          400
        ELSE
          70

```



```

300 GET(0,0)-(319,39),T
310 FOR I=40 TO 199 STEP 40
320   GET(0,I)-(319,I+39),M
330   PUT(0,I-40),M,PSET
340 NEXT
350 PUT(0,160),T,PSET:GOTO 70
400 GET(0,160)-(319,199),T
410 FOR I=120 TO 0 STEP -40
420   GET(0,I)-(319,I+39),M
430   PUT(0,I+40),M,PSET
440 NEXT
450 PUT(0,0),T,PSET:GOTO 70

```

PROGRAMME 9.20

La combinaison des deux mouvements décrits ci-dessus donne un écran qui boucle horizontalement et verticalement. Le meilleur moyen de concevoir ce mouvement est un tore, qui est une figure en forme d'anneau avec une surface continue, à la fois horizontalement et verticalement. La figure 9.6 montre la transformation de l'écran en tore.

Le programme 9.21 déplace l'écran dans n'importe quelle direction, avec bouclage, en utilisant les quatre touches numériques du bloc pour le déplacement du curseur ; qui sont 8 pour aller en haut, 2 pour le bas, 4 pour la gauche et 6 pour la droite.

```

0 ' ** 9-21 : Toroidal wrap around **
5 COL=1: DIM M(900),T(900)
10 SCREEN 1:CLS
20 FOR I=0 TO 199 STEP 5
30   LINE(0,0)-(159,I),COL
40   LINE(319,199)-(160,I),COL
50   COL=COL+1:
    IF COL=4
      THEN
        COL=1
60 NEXT
70 W$=INPUT$(1)
80 IF W$="4"
    THEN
      100
    ELSE
      IF W$="6"
        THEN
          200
        ELSE
          IF W$="8"
            THEN
              300
            ELSE
              IF W$="2"
                THEN
                  400
                ELSE
                  SOUND 1000,1:GOTO 70
100 GET(0,0)-(39,199),T
110 FOR I=40 TO 319 STEP 40
120   GET(I,0)-(I+39,199),M
130   PUT(I-40,0),M,PSET

```

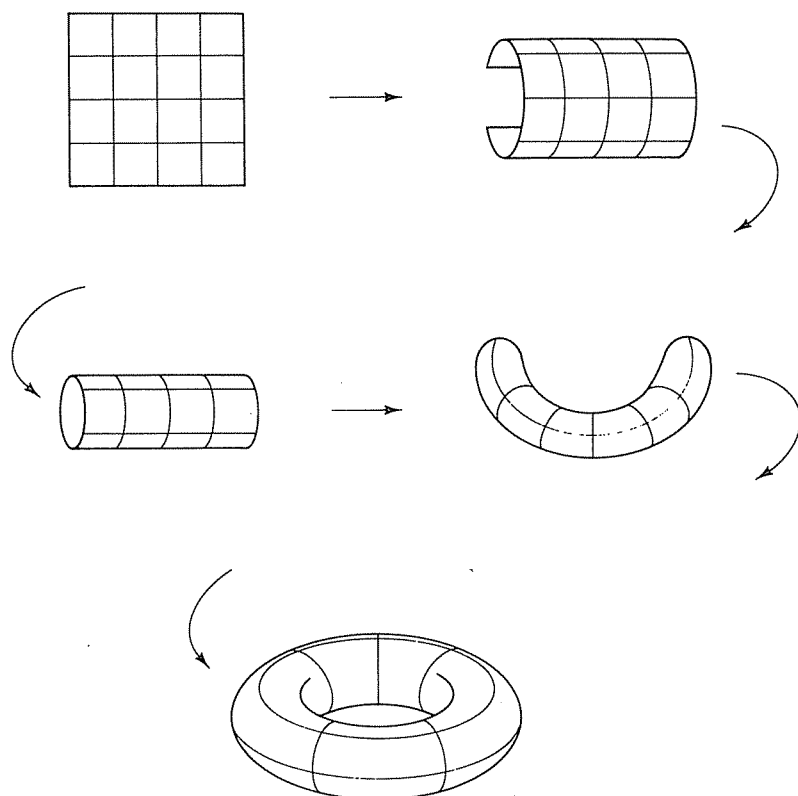


Fig. 9.6

```

140 NEXT
150 PUT(280,0),T,PSET:GOTO 70
200 GET(280,0)-(319,199),T
210 FOR I=240 TO 0 STEP -40
220   GET(I,0)-(I+39,199),M
230   PUT(I+40,0),M,PSET
240 NEXT
250 PUT(0,0),T,PSET:GOTO 70
300 GET(0,0)-(319,39),T
310 FOR I=40 TO 199 STEP 40
320   GET(0,I)-(319,I+39),M
330   PUT(0,I-40),M,PSET
340 NEXT
350 PUT(0,160),T,PSET:GOTO 70
400 GET(0,160)-(319,199),T
410 FOR I=120 TO 0 STEP -40
420   GET(0,I)-(319,I+39),M
430   PUT(0,I+40),M,PSET
440 NEXT
450 PUT(0,0),T,PSET:GOTO 70

```

PROGRAMME 9.21

9.11 DISQUE ET ECRAN

Il est évident, comme on l'a vu dans les paragraphes précédents que la paire GET et PUT permet une très grande souplesse parce que des parties de l'écran peuvent être temporairement sauvegardées et rétablies plus tard. C'est épatant, par exemple, de pouvoir sauvegarder la totalité de l'écran, ou du moins la partie qui va être immédiatement affectée, de telle sorte qu'on puisse oublier tout changement non voulu en retournant à l'état initial, comme si on disait à l'ordinateur « Pouvez-vous oublier ce que je viens de vous dire ? » Malheureusement les restrictions de mémoire limitent la quantité que l'on peut stocker pour une utilisation ultérieure. Ce serait fantastique, par exemple, si un programme sauvegardait la totalité de l'écran périodiquement de telle sorte qu'on pourrait avoir non seulement l'écran précédent, mais aussi celui d'avant et ainsi de suite jusqu'au premier.

On a besoin de 16 koctets pour mémoriser chaque écran, et avec quatre on épuiserait la capacité de l'ordinateur ⁶. Heureusement il y a une autre façon de mémoriser les données, sur les dispositifs de mémorisation externes. Il y a les unités de disques souples, les disques durs, différents types de bandes, des pseudo disques semi-conducteurs, etc. Comme les processus de mémorisation et de rétablissement sont pris en charge directement par le système d'exploitation (MS DOS), nous lui laisserons tout le travail pénible et parlerons des disques en tant que support de mémoire externe.

9.11.1 Sauvegarde de l'écran

La paire d'instructions BSAVE et BLOAD sont très utiles pour mémoriser les écrans graphiques. Elles sont supposées transférer n'importe quelle partie spécifiée de la mémoire centrale sur et à partir des disques. Les syntaxes des deux sont BSAVE « *nom-de-fichier* », *début*, *longueur* et BLOAD « *nom-de-fichier* », *adresse*. Une discussion sur le partage de la mémoire utilisée par l'IBM PC dépasse le champ de ce livre ⁷; nous mentionnerons seulement ici que pour accéder directement à la mémoire graphique il est nécessaire de positionner un pointeur à l'adresse mémoire correspondante. Ceci est réalisé par l'instruction DEF SEG=&HB800. De cette façon, les 16K (16 834 octets) du tableau graphique sont accessibles par n'importe laquelle des instructions relatives à des emplacements mémoire (PEEK, POKE, BSAVE et BLOAD). L'instruction BSAVE, « *nom-de-fichier* », 0, 16384 sauvegarde tout l'écran graphique sur le disque avec le nom « *nom-de-fichier* ». L'instruction BLOAD, « *nom-de-fichier* » charge sur l'écran les données précédemment sauvegardées par BSAVE. Bien que l'instruction BLOAD comprenne une adresse optionnelle, en général vous voudrez remettre l'information de l'écran à l'endroit où elle a été prise et pas ailleurs.

Pour montrer l'utilisation de ces deux instructions, le programme 9.22 sauvegarde et charge le contenu de l'écran sur et à partir d'un disque. Il peut être fusionné à un programme de création graphique. Pour éviter d'affecter l'écran, le choix d'une des trois options est reconnue par un son : un son grave (obtenu après avoir tapé le « S » pour « save ») quand l'écran va être sauvegardé, un son aigu (obtenu après avoir tapé le « L » pour « load ») quand l'écran va être chargé et un son medium (obtenu par « Q » pour « quit ») quand le programme doit être terminé. Le nom de fichier utilisé est toujours « SCREEN » (écran).

```

0  '°° 9-22 : Save and load screen °°
10 DEF SEG=&HB800
20 W$=INPUT$(1)
30 IF W$="S"OR W$="s"
    THEN
        SOUND 5000,5:BSAVE"SCREEN",0,16384:GOTO 20
40 IF W$="L"OR W$="l"
    THEN
        SOUND 100,5:BLOAD"SCREEN":GOTO 20
50 IF W$="Q"OR W$="q"
    THEN
        SOUND 1000,5:END
60 GOTO 20

```

PROGRAMME 9.22

Notez que le programme 9.22 n'efface pas l'écran ni ne fixe pas le mode graphique. S'il le faisait, cela détruirait le contenu de l'écran qui doit être sauvegardé. C'est la responsabilité de l'utilisateur que de fixer le mode avant de fusionner les deux programmes. Essayez de charger un graphique quand l'ordinateur est en mode texte (réglé avec SCREEN 0).

9.11.2 Sauvegarde de parties de l'écran

Pour ne sauvegarder qu'une partie de l'écran, il n'est pas pratique d'utiliser les instructions BSAVE et BLOAD en prenant les données directement dans la zone mémoire graphique. A cause de la correspondance non séquentielle⁸ entre l'écran et sa mémoire, il est difficile de mémoriser une partie de celui-ci. Une autre méthode consiste à mémoriser une partie de l'écran dans un tableau avec GET, et ensuite sauvegarder les données sur disque à partir du tableau. La façon la plus directe de réaliser cela est de sauvegarder la valeur de chaque nombre du tableau. La première partie du programme 9.23 sauvegarde le petit visage créé par les lignes 20 à 50 dans le fichier séquentiel « FACE ». La deuxième partie à laquelle on accède par l'instruction RUN 500, efface l'écran et recharge les données

dans le tableau pour restituer la forme sur l'écran.

```

0  '** 9-23 : Copy window to disk **
10 SCREEN 1:CLS:DIM M%(37)
20 FOR I=1 TO 62
30   READ X,Y
40   PSET(100+X,100+Y)
50 NEXT
80 GET(100,100)-(115,117),M%
90 OPEN"O",1,"FACEDATA"
100 FOR I=0 TO 37
110   PRINT#1,M%(I)
120 NEXT
130 CLOSE:END
500 DIM M%(37):SCREEN 1:CLS
510 OPEN"I",1,"FACEDATA"
520 FOR I=0 TO 37
530   INPUT#1,M%(I)
540 NEXT
550 PUT(100,100),M%
560 END
1000 DATA 11,0,10,0,9,0,8,0,7,0,6,0,5,
          1,4,1,3,2,3,3,2,4,2,5,2,6,1,
          7,1,8,0,9,0,10
1010 DATA 1,10,2,10,2,11,2,12,3,12,4,
          12,5,11,3,13,3,14,3,15,4,15,
          5,15,6,15,6,16
1020 DATA 6,17,4,6,5,6,6,6,4,7,5,7
1030 DATA 12,0,13,1,14,2,14,3,14,4,15,
          5,15,6,15,7,15,8,14,9,14,10,
          14,11,14,12
1040 DATA 13,12,13,13,12,14,12,15,12,
          16,12,17,9,7,10,6,11,6,12,7,
          11,8,11,9

```

PROGRAMME 9.23

L'utilisation de fichiers séquentiels sur disques a trois inconvénients : premièrement on gaspille de la place sur le disque ; deuxièmement c'est très lent ; et troisièmement, ce qui est le pire, on ne peut utiliser que des nombres entiers parce que la représentation séquentielle de nombres en virgule flottante sur disques n'est pas toujours précise⁹.

La fonction VARPTR retourne l'adresse de n'importe quelle variable en mémoire. Par exemple, après l'exécution de ADDR=VARPTR(I), ADDR contient l'adresse du contenu de la variable I¹⁰. L'exécution de l'instruction ADDR=VARPTR(M(0)) affecte à ADDR l'adresse où est placé le premier élément du tableau M en mémoire. Les tableaux sont mémorisés séquentiellement dans des emplacements de mémoire contigus, ainsi on peut être sûr que le reste du tableau est placé à partir de ADDR et au-delà. On peut calculer facilement la longueur de la représentation interne d'une partie de l'écran avec l'équation 9.1, et avec celle-ci nous avons toute l'information nécessaire pour sauvegarder une partie de l'écran sur disque. Le processus complet exige les étapes suivantes :

1. Placer (GET) la partie de l'écran dans un tableau.
2. Trouver l'adresse du premier élément du tableau avec la fonction VARPTR.
3. Trouver la longueur des données dans le tableau avec l'équation 9.1.
4. Sauvegarder (BSAVE) la partie du mémoire qui commence au premier élément (0ème ou 1er) du tableau et a la longueur trouvée à l'étape 3.

Le processus pour restituer la figure est :

1. Trouver l'adresse du premier élément du tableau.
2. Charger (BLOAD) le fichier commençant à l'adresse trouvée à l'étape 1.
3. Ecrire (PUT) le tableau sur l'écran.

La première partie du programme 9.24 sauvegarde la figure créée par les lignes 20 à 24 dans le fichier « SECTION ». La deuxième partie, à laquelle on accède par l'instruction RUN 500, charge les données et remplace la figure sur l'écran.

```

0 '** 9-24 : Save binary file **
10 SCREEN 1:CLS:COLOR 0,0
15 DIM M(938)
20 FOR I=0 TO 60 STEP 10
30   LINE(60,I)-(60+I,60)
32   LINE(60,I)-(60-I,60)
34   LINE(60,120-I)-(60-I,60)
36   LINE(60,120-I)-(60+I,60)
40 NEXT
50 GET(0,0)-(120,120),M
60 DEF SEG
70 BSAVE"TEMPDATA",VARPTR(M(0)),3755
80 END
500 SCREEN 1:CLS
510 DIM M(938)
520 DEF SEG
530 BLOAD"TEMPDATA",VARPTR(M(0))
540 PUT(50,50),M
550 W$=INPUT$(1)

```

PROGRAMME 9.24

Le programme 9.25 utilise la même technique que le programme 9.8 pour isoler la partie de l'écran qui sera sauvegardée sous le nom « SECTION ». La deuxième partie restitue la figure sur l'écran. Notez que la dimension de M laisse assez de place pour tout l'écran, puisqu'on ne connaît pas la taille de la forme sauvegardée à l'origine.

```

0 '** 9-25 : Select and save window **
10 X1=-1:Y1=-1:X2=320:Y2=200
15 DIM M(4000):D=1:COL=3
20 W$=INPUT$(1)

```

```

30 IF W$="+"
    THEN
        D=10:GOTO 20
40 IF W$="-"
    THEN
        D=1:GOTO 20
50 IF W$="B"
    THEN
        Y1=Y1+D:LINE(X1,Y1)-(X2,Y1),COL:
        GOTO 20
60 IF W$="2"
    THEN
        Y2=Y2-D:LINE(X1,Y2)-(X2,Y2),COL:
        GOTO 20
70 IF W$="4"
    THEN
        X1=X1+D:LINE(X1,Y1)-(X1,Y2),COL:
        GOTO 20
80 IF W$="6"
    THEN
        X2=X2-D:LINE(X2,Y1)-(X2,Y2),COL:
        GOTO 20
90 IF W$<>"C"AND W$<>"c"
    THEN
        100
93 COL=COL+1:
    IF COL=4
        THEN
            COL=1
95 LINE(X1,Y1)-(X2,Y2),COL,B:GOTO 20
100 IF W$=CHR$(13)
    THEN
        200
110 SOUND 1000,.1:GOTO 20
200 X1=X1+1:X2=X2-1:Y1=Y1+1:Y2=Y2-1
210 GET(X1,Y1)-(X2,Y2),M
220 SIZE=4+INT(((X2-X1+1)*2+7)/8)*(Y2-Y1+1)
230 BSAVE"SECTION",VARPTR(M(0)),SIZE
240 END
500 SCREEN 1:CLS
510 DIM M(4000)
520 BLOAD"SECTION",VARPTR(M(0))
530 PUT(0,0),M
540 W$=INPUT$(1)

```

PROGRAMME 9.25

9.12 LA PALETTE ETENDUE AVEC GET ET PUT

Un des problèmes de la palette étendue est que c'est très lent. Pour remplir un petit carré de quelques pixels, le programme doit faire beaucoup de comparaisons, et ce n'est pas pratique pour la création interactive de graphiques dans laquelle le temps est essentiel. Des programmes comme les programmes 9.9 et 9.11 par exemple deviennent trop lourds quand on leur ajoute la palette étendue. Heureusement il existe une manière d'accélérer le processus à l'aide de GET et PUT. Ce que nous allons faire c'est

avoir un tableau pour chaque couleur de telle manière que chaque fois qu'on aura besoin d'un bloc particulier d'une certaine taille on n'aie pas à avoir à le tracer point par point, mais avec une simple instruction. Le programme 9.26 utilise le sous-programme 2000 du programme 4.16 pour produire les 20 couleurs différentes de la palette étendue, couleurs qui sont copiées dans 20 tableaux, de C1 à C20. Les quatre mouvements du curseur sont dirigés par les quatre touches habituelles (8, 2, 4 et 6) et le changement de couleur par la touche « C ».

```

0  ** 9-26 : Sketch board with palette **
20 DEFINT A-Z:N=3
30 DIM A(20,4):DIM C1(N),C2(N),C3(N),C4(N),
    C5(N),C6(N),C7(N),C8(N),C9(N),C10(N),
    C11(N),C12(N),C13(N),C14(N),C15(N),
    C16(N),C17(N),C18(N),C19(N),C20(N)
40 FOR I=1 TO 20
50   FOR J=1 TO 4
60     READ A(I,J)
70   NEXT
80 NEXT
90 SCREEN 1:COLOR 1,0:CLS
100 FOR COL=1 TO 20
110   FOR Q=0 TO 2 STEP 2
120     FOR W=0 TO 2 STEP 2
130       GOSUB 2000
140     NEXT
150   NEXT
160   ON COL GOTO
      170,180,190,200,210,220,230,240,
      250,260,270,280,290,300,310,320,
      330,340,350,360
170   GET(0,0)-(3,3),C1:GOTO 370
180   GET(0,0)-(3,3),C2:GOTO 370
190   GET(0,0)-(3,3),C3:GOTO 370
200   GET(0,0)-(3,3),C4:GOTO 370
210   GET(0,0)-(3,3),C5:GOTO 370
220   GET(0,0)-(3,3),C6:GOTO 370
230   GET(0,0)-(3,3),C7:GOTO 370
240   GET(0,0)-(3,3),C8:GOTO 370
250   GET(0,0)-(3,3),C9:GOTO 370
260   GET(0,0)-(3,3),C10:GOTO 370
270   GET(0,0)-(3,3),C11:GOTO 370
280   GET(0,0)-(3,3),C12:GOTO 370
290   GET(0,0)-(3,3),C13:GOTO 370
300   GET(0,0)-(3,3),C14:GOTO 370
310   GET(0,0)-(3,3),C15:GOTO 370
320   GET(0,0)-(3,3),C16:GOTO 370
330   GET(0,0)-(3,3),C17:GOTO 370
340   GET(0,0)-(3,3),C18:GOTO 370
350   GET(0,0)-(3,3),C19:GOTO 370
360   GET(0,0)-(3,3),C20:GOTO 370
370 NEXT:CLS
380 X=160:Y=100:COL=1
390 W$=INPUT$(1)
400 IF W$<>"C"
    THEN
      430
410 COL=COL+1:

```



```

IF COL=21
  THEN
    COL=1
420 GOTO 480
430 IF W$="4"
  THEN
    IF X>=4
    THEN
      X=X-4:GOTO 480
    ELSE
      BEEP:GOTO 390
440 IF W$="6"
  THEN
    IF X<=315
    THEN
      X=X+4:GOTO 480
    ELSE
      BEEP:GOTO 390
450 IF W$="8"
  THEN
    IF Y>=4
    THEN
      Y=Y-4:GOTO 480
    ELSE
      BEEP:GOTO 390
460 IF W$="2"
  THEN
    IF Y<=195
    THEN
      Y=Y+4:GOTO 480
    ELSE
      BEEP:GOTO 390
470 BEEP:GOTO 390
480 ON COL GOTO
      490,500,510,520,530,540,550,560,
      570,580,590,600,610,620,630,640,
      650,660,670,680
490 PUT(X,Y),C1,PSET:GOTO 690
500 PUT(X,Y),C2,PSET:GOTO 690
510 PUT(X,Y),C3,PSET:GOTO 690
520 PUT(X,Y),C4,PSET:GOTO 690
530 PUT(X,Y),C5,PSET:GOTO 690
540 PUT(X,Y),C6,PSET:GOTO 690
550 PUT(X,Y),C7,PSET:GOTO 690
560 PUT(X,Y),C8,PSET:GOTO 690
570 PUT(X,Y),C9,PSET:GOTO 690
580 PUT(X,Y),C10,PSET:GOTO 690
590 PUT(X,Y),C11,PSET:GOTO 690
600 PUT(X,Y),C12,PSET:GOTO 690
610 PUT(X,Y),C13,PSET:GOTO 690
620 PUT(X,Y),C14,PSET:GOTO 690
630 PUT(X,Y),C15,PSET:GOTO 690
640 PUT(X,Y),C16,PSET:GOTO 690
650 PUT(X,Y),C17,PSET:GOTO 690
660 PUT(X,Y),C18,PSET:GOTO 690
670 PUT(X,Y),C19,PSET:GOTO 690
680 PUT(X,Y),C20,PSET:GOTO 690
690 GOTO 390
700 END
2000 ' Draw basic 2*2 block
2010 PSET(Q,W),A(COL,1):PSET(Q+1,W),A(COL,2)

```

```

2020 PSET (Q,W+1),A(COL,3):PSET (Q+1,W+1),A(COL,4)
2030 RETURN
2040 DATA 1,0,0,0,0,1,1,0,1,1,0,1,1,1,1,1,2,0,0,
          0,0,2,2,0,2,2,0,2,2,2,2,2,3,0,0,0,0,3,
          3,0,3,3,0,3,3,3,3,2,1,1,1,1,2,2,1,2,
          2,1,2,3,2,2,2,2,3,3,2,3,3,2,3,0,0,0,0,
          0,1,2,3

```

PROGRAMME 9.26

Dans le programme 9.27 nous avons étendu le curseur du programme 9.26 à trois tailles différentes : les carrés de 2 par 2 sont assez petits pour être dessinés directement par le sous-programme 2000, ceux de 4 par 4 sont dessinés avec les tableaux et ceux de 8 par 8 sont dessinés par quatre des blocs rangés dans les tableaux. La touche pour modifier la dimension du curseur est le « S ». Notez que pour éviter d'écrire les tableaux à l'extérieur de l'écran, il est nécessaire d'arrêter le mouvement horizontal à 311 et le vertical à 191, bien que les curseurs les plus petits puissent aller plus loin. Si on changeait la dimension du petit curseur, le bloc le plus gros pourrait essayer d'écrire (PUT) le contenu du tableau sur une position non autorisée.

```

0  '*** 9-27 : Sketch board with **
10  '* palette & variable cursor size *
20  DEFINT A-Z:N=3
30  DIM A(20,4):DIM C1(N),C2(N),C3(N),C4(N),
      C5(N),C6(N),C7(N),C8(N),C9(N),C10(N),
      C11(N),C12(N),C13(N),C14(N),C15(N),
      C16(N),C17(N),C18(N),C19(N),C20(N)
40  FOR I=1 TO 20
50    FOR J=1 TO 4
60      READ A(I,J)
70    NEXT
80  NEXT
90  SCREEN 1:COLOR 1,0:CLS
100  FOR COL=1 TO 20
110    FOR Q=0 TO 2 STEP 2
120      FOR W=0 TO 2 STEP 2
130        GOSUB 2000
140      NEXT
150    NEXT
160    ON COL GOTO
          170,180,190,200,210,220,230,240,
          250,260,270,280,290,300,310,320,
          330,340,350,360
170  GET(0,0)-(3,3),C1:GOTO 370
180  GET(0,0)-(3,3),C2:GOTO 370
190  GET(0,0)-(3,3),C3:GOTO 370
200  GET(0,0)-(3,3),C4:GOTO 370
210  GET(0,0)-(3,3),C5:GOTO 370
220  GET(0,0)-(3,3),C6:GOTO 370
230  GET(0,0)-(3,3),C7:GOTO 370
240  GET(0,0)-(3,3),C8:GOTO 370
250  GET(0,0)-(3,3),C9:GOTO 370
260  GET(0,0)-(3,3),C10:GOTO 370
270  GET(0,0)-(3,3),C11:GOTO 370

```

```

280 GET(0,0)-(3,3),C12:GOTO 370
290 GET(0,0)-(3,3),C13:GOTO 370
300 GET(0,0)-(3,3),C14:GOTO 370
310 GET(0,0)-(3,3),C15:GOTO 370
320 GET(0,0)-(3,3),C16:GOTO 370
330 GET(0,0)-(3,3),C17:GOTO 370
340 GET(0,0)-(3,3),C18:GOTO 370
350 GET(0,0)-(3,3),C19:GOTO 370
360 GET(0,0)-(3,3),C20:GOTO 370
370 NEXT:CLS
380 XX=160:YY=100:COL=1:SIZE=4:GOTO 471
390 W$=INPUT$(1)
400 IF W$<>"c"AND W$<>"C"
    THEN
        424
410 COL=COL+1:
    IF COL=21
        THEN
            COL=1
420 GOTO 471
424 IF W$<>"s"AND W$<>"S"
    THEN
        430
425 SIZE=2*SIZE:
    IF SIZE=16
        THEN
            SIZE=2
426 GOTO 471
430 IF W$="4"
    THEN
        IF XX-SIZE>=0
            THEN
                XX=XX-SIZE:GOTO 471
            ELSE
                BEEP:GOTO 390
440 IF W$="6"
    THEN
        IF XX+SIZE<=311
            THEN
                XX=XX+SIZE:GOTO 471
            ELSE
                BEEP:GOTO 390
450 IF W$="8"
    THEN
        IF YY-SIZE>=0
            THEN
                YY=YY-SIZE:GOTO 471
            ELSE
                BEEP:GOTO 390
460 IF W$="2"
    THEN
        IF YY+SIZE<=191
            THEN
                YY=YY+SIZE:GOTO 471
            ELSE
                BEEP:GOTO 390
470 BEEP:GOTO 390
471 IF SIZE=2
    THEN
        472

```

```

ELSE
  IF SIZE=4
    THEN
      473
    ELSE
      IF SIZE=8
        THEN
          475
        ELSE
          BEEP:STOP
472 Q=XX:W=YY:GOSUB 2000:GOTO 390
473 X=XX:Y=YY:GOSUB 480:GOTO 390
475 FOR X=XX TO XX+4 STEP 4
476   FOR Y=YY TO YY+4 STEP 4
477     GOSUB 480
478   NEXT
479 NEXT:GOTO 390
480 ON COL GOTO
      490,500,510,520,530,540,550,560,
      570,580,590,600,610,620,630,640,
      650,660,670,680
490 PUT(X,Y),C1,PSET:GOTO 690
500 PUT(X,Y),C2,PSET:GOTO 690
510 PUT(X,Y),C3,PSET:GOTO 690
520 PUT(X,Y),C4,PSET:GOTO 690
530 PUT(X,Y),C5,PSET:GOTO 690
540 PUT(X,Y),C6,PSET:GOTO 690
550 PUT(X,Y),C7,PSET:GOTO 690
560 PUT(X,Y),C8,PSET:GOTO 690
570 PUT(X,Y),C9,PSET:GOTO 690
580 PUT(X,Y),C10,PSET:GOTO 690
590 PUT(X,Y),C11,PSET:GOTO 690
600 PUT(X,Y),C12,PSET:GOTO 690
610 PUT(X,Y),C13,PSET:GOTO 690
620 PUT(X,Y),C14,PSET:GOTO 690
630 PUT(X,Y),C15,PSET:GOTO 690
640 PUT(X,Y),C16,PSET:GOTO 690
650 PUT(X,Y),C17,PSET:GOTO 690
660 PUT(X,Y),C18,PSET:GOTO 690
670 PUT(X,Y),C19,PSET:GOTO 690
680 PUT(X,Y),C20,PSET:GOTO 690
690 RETURN
700 END
2000 ' Draw basic 2*2 block
2010 PSET(Q,W),A(COL,1):PSET(Q+1,W),A(COL,2)
2020 PSET(Q,W+1),A(COL,3):PSET(Q+1,W+1),A(COL,4)
2030 RETURN
2040 DATA 1,0,0,0,0,1,1,0,1,1,0,1,1,1,1,2,0,0,
      0,0,2,2,0,2,2,0,2,2,2,2,2,3,0,0,0,3,
      3,0,3,3,0,3,3,3,3,2,1,1,1,1,2,2,1,2,
      2,1,2,3,2,2,2,2,3,3,2,3,3,2,3,0,0,0,0,
      0,1,2,3

```

PROGRAMME 9.27

9.14 LA COULEUR ALEATOIRE

Les quatre premiers octets d'un tableau où une partie de l'écran est mémorisée sont utilisés pour indiquer les dimensions horizontale et verticale de la première zone mémorisée. Ces quatre octets ne doivent pas être touchés, ou alors la dimension du rectangle d'origine sera perdue. La suite du tableau contient les données. Si on change ces variables, la figure originale ne sera pas préservée, mais l'instruction PUT fonctionnera toujours correctement. Si nous ventilons des nombres au hasard dans ces emplacements, le résultat sera un bloc de couleurs aléatoires. Le bout de programme 9.28 modifie le contenu du tableau C1. Si on ajoute cette ligne au programme 9.27, la couleur #1 de la palette étendue sera une « couleur aléatoire », différente chaque fois. Comme seul le bloc de 4 par 4 est changé, celui de 2 par 2 (tracé par le sous-programme 2000) sera préservé, et celui de 8 par 8 montrera quatre blocs identiques colorés aléatoirement.

```
390 W$=INKEY$:
    IF W$=" "
    THEN
        C1(2+1.49°RND)=32767°RND:GOTO 390
```

PROGRAMME 9.28

9.15 LE CURSEUR VISEUR

Quand on crée des figures interactivement, il est toujours utile que le curseur, non seulement montre où le prochain point ou bloc sera tracé, mais aussi sa relation avec les formes avoisinantes. Une manière de montrer cette relation est un curseur viseur, qui est une croix avec un trou en son centre et qui aide à voir si le curseur est aligné avec d'autres points ou lignes. Comme tous les curseurs, celui-ci ne doit pas affecter le fond. Une façon d'y parvenir sera de mémoriser à chaque fois la partie de l'écran où la croix va apparaître. Une autre méthode que nous avons choisie consiste à ranger la croix dans un tableau et à la placer (PUT) sur l'écran en mode XOR de telle sorte que le fond soit restitué chaque fois que le tableau est affiché (PUT) une seconde fois.

Dans le programme 9.29, la croix est affichée dans le coin en haut à gauche de l'écran et copiée dans le tableau CUR. La lecture est faite avec la fonction INKEY\$ et on utilise le compteur C pour garder une trace du temps passé depuis la dernière fois où CUR a été placé sur l'écran. FL indique si le curseur est allumé ou pas : s'il l'est, il doit être éteint avant d'être placé à un nouvel endroit.

```

0  '** 9-29 : Cross hair cursor **
10 SCREEN 1:CLS:DIM M(100)
20 LINE(0,4)-(8,4)
30 LINE(4,0)-(4,8)
40 LINE(3,4)-(5,4),0
50 LINE(4,3)-(4,5),0
60 GET(0,0)-(8,8),M:CLS
70 RADIUS=90:F=1.745329E-02
80 FOR C=2 TO 2 STEP -1
90 FOR ANGLE=0 TO 360 STEP 5
100  X=160+RADIUS*COS(ANGLE*F)
110  Y=100+RADIUS*SIN(ANGLE*F)
120  IF ANGLE>0
      THEN
        LINE(PX,PY)-(X,Y),C
130  PX=X:PY=Y
140  IF ANGLE MOD 10=0
      THEN
        LINE(X,Y)-(160,100),C
150 NEXT
160 NEXT
200 W$=INKEY$:
    IF W$<>""
      THEN
        300
210 C=C+1:
    IF C<30
      THEN
        200
220 C=0
230 FL=NOT FL
240 PUT(X,Y),M
250 GOTO 200
300 IF W$="2"
      THEN
        GOSUB 1000:Y=Y+5:GOTO 200
310 IF W$="8"
      THEN
        GOSUB 1000:Y=Y-5:GOTO 200
320 IF W$="4"
      THEN
        GOSUB 1000:X=X-5:GOTO 200
330 IF W$="6"
      THEN
        GOSUB 1000:X=X+5:GOTO 200
340 BEEP:GOTO 200
1000 IF FL
      THEN
        PUT(X,Y),M
1010 FL=0:RETURN

```

PROGRAMME 9.29

9.16 LA COMBINAISON DE FIGURES

Quelquefois la création d'une image doit être divisée en deux processus ou plus, pour chacun desquels on a besoin de la totalité de l'écran. En

utilisant GET, des figures intermédiaires peuvent être mémorisées tandis que la suivante est créée. Quand tout le processus est terminé, les parties individuelles peuvent être assemblées pour produire l'image finale. Si on n'a besoin que de deux parties et qu'on dispose de suffisamment de mémoire, la première peut être sauvegardée dans un tableau, l'écran peut être effacé, la seconde partie dessinée, et la première combinée avec la seconde en écrivant (PUT) sous le mode OR. Par exemple la création de la figure 11.1.6 exige cette technique.

Si on a besoin de plus de deux images, il y a deux manières possibles de les manipuler.

Dans la première, la première figure est créée et rangée dans un tableau, la deuxième est dessinée, et combinée avec la première. La figure résultante est de nouveau rangée dans un tableau et ce processus peut continuer avec la troisième qui sera plus tard combinée avec les deux précédentes. Avec cette méthode on peut combiner n'importe quel nombre d'images.

Dans la seconde méthode nous supposons que certaines figures intermédiaires ne peuvent pas être combinées avec les précédentes parce qu'elles contiennent des informations nécessaires pour produire d'autres figures, informations qui seraient perdues si on les combinait avec d'autres. Dans ce cas les images sont créées, rangées dans un tableau et sauvegardées sur disque. Quand les étapes intermédiaires sont terminées, les figures peuvent être lues à partir du disque dans le tableau et écrites (PUT) sur l'écran sous le mode OR. Ceci ne peut pas être réalisé quand l'image est sauvegardée par BSAVE parce que lorsqu'on la charge (BLOAD) le contenu de l'écran est complètement détruit.

9.17 LE PASSAGE D'UN MODE A L'AUTRE

Quand un graphique a été créé dans un mode précis (moyenne résolution par exemple), un certain nombre de modifications sont nécessaires pour le transcrire sous un autre mode (haute résolution dans cet exemple) : il faut en effet compenser la différence de taille des pixels et de l'écran. Quand la totalité de l'écran, ou une partie de ce dernier est mémorisée dans un tableau, on peut changer le mode sans perdre l'information sur la forme. Si on écrit (PUT) le tableau sur l'écran dans le nouveau mode, les proportions seront préservées : si on passe de la moyenne à la haute résolution les lignes auront l'air plus fines et, dans le cas contraire, certaines parties des lignes blanches seront interprétées comme une information sur la couleur, donc des parties du graphique seront colorées.

Essayez la séquence suivante : réglez en moyenne résolution avec SCREEN 1, affichez le répertoire du disque en cours d'enregistrement avec FILES, réservez de la mémoire avec DIM M(4000) et saisissez l'écran avec GET(0,0)-(319,199),M. Maintenant changez de mode avec SCREEN 2 et finalement affichez le tableau sur l'écran avec PUT(0,0),M,PSET. :

le contenu du répertoire sera affiché avec des caractères en moyenne résolution alors qu'on est en haute résolution ! Bien sûr le transfert en sens inverse est également possible et aussi facile.

EXERCICES

1. Ajouter un curseur clignotant aux programmes de table traçante.
2. La couleur 19 de la table traçante est la couleur du fond. Si on déplace le curseur sur des zones qui ont déjà été colorées, elle devient visible, sinon on ne la voit pas. Modifier le programme 9.27 de telle sorte que lorsque la couleur 19 est sélectionnée, le curseur clignotant soit toujours visible (avec n'importe laquelle des couleurs de fond) mais clignote plus rapidement pour le distinguer des autres.
3. Ajouter une « couleur invisible » à la table traçante du programme 9.27 de telle sorte que le curseur puisse se déplacer sans détruire le fond. Comme le curseur de l'exercice 2, celui-ci doit clignoter plus rapidement pour qu'il soit différencié du reste.
4. Ecrire un programme dans lequel l'écran boucle comme dans le programme 9.21, mais avec une bande « cachée » dans chaque direction. Par exemple quand l'écran est décalé d'un côté, une des sections devra disparaître et de l'autre côté une nouvelle, jusqu'alors cachée, apparaître.
5. Modifier le programme 9.11 de sorte que la zone minimum nécessaire soit sauvegardée et non pas l'écran entier. Ceci accélérera le programme d'une manière étonnante.
6. Ecrire un programme dans lequel des lignes du curseur viseur s'étendent en travers de tout l'écran. Quand le curseur se déplace, *ne* copiez *pas* l'écran en entier dans un tableau, mais seulement les zones couvertes par les lignes.
7. Modifier le programme de couleur aléatoire (9.27 et 9.28 combinés) de telle sorte que toutes les dimensions de curseurs apparaissent réellement combinées de façon aléatoire.

NOTES

1. C'est la même zone que celle couverte par le rectangle produit par l'instruction `LINE(0,0)-(100,100),3,B`, y compris le cadre.
2. Si `OPTION BASE 1` a été exécutée, l'élément 0 n'existera pas et toute tentative pour l'utiliser provoquera une erreur d'appel de fonction non autorisé.

3. C'est la ligne la plus longue ayant ses points limites dans l'écran.
4. Quand la touche « Num lock » n'est pas active, les touches fléchées (appelées touches de déplacement) produisent un code à deux caractères qui peut être lu avec INKEY\$. Ici nous utilisons les chiffres du pavé numérique.
5. La fonction SCREEN retourne le code ASCII du caractère dont les coordonnées sont passées comme paramètres. Une autre manière de déplacer l'écran est de prendre le code ASCII de chaque caractère et de l'imprimer à son nouvel emplacement avec la fonction CHR\$.
6. Bien que l'IBM PC puisse avoir beaucoup plus que 64K de mémoire, toutes ses versions de BASIC ne peuvent accéder qu'à 64K.
7. Une description détaillée des adresses des deux adaptateurs d'écran utilisés dans le PC, de même que celles des instructions BSAVE et BLOAD peut être trouvée dans le livre « FANCY PROGRAMMING WITH IBM PC BASIC » du même auteur.
8. Pour connaître l'organisation interne de la zone mémoire réservée au graphique, exécutez le programme suivant :
10 SCREEN 1:DEF SEG=&H800
20 FOR I=0 TO 16383:POKE I,255:NEXT
9. Quand les nombres en virgule flottante sont affichés sous une forme lisible, une conversion est faite entre la représentation binaire externe et la représentation décimale que nous utilisons. Cette conversion conduit quelquefois à quelques imprécisions intolérables dans le cas des images.
10. A cause de l'utilisation d'instructions d'accès direct en mémoire, il est nécessaire d'utiliser l'instruction DEF SEG, ainsi la donnée est prise à partir de la zone où résident les variables.

10

L'animation

L'animation sur un ordinateur est un des moyens les plus dynamiques pour afficher l'information. Par le mouvement, les objets et les formes acquièrent une vie propre et un impact difficiles à réaliser avec des images fixes. Par exemple, si les jeux vidéos sont excitants, c'est dû en grande partie à leur animation rapide. Bien qu'il y ait un charme unique dans les jeux de hasard dans lequel le joueur doit s'embarquer dans de longs dialogues avec l'ordinateur, ces jeux ne peuvent pas rivaliser avec la véritable émotion d'un jeu d'arcade avec une animation rapide et précise.

Comme dans les films ou la télévision, dans l'animation sur ordinateur l'affichage rapide d'images immobiles qui changent de position ou de forme donne l'illusion du mouvement. BASIC possède des outils puissants pour afficher des images à une grande vitesse. Dans ce chapitre nous étudierons ces outils et quelques manières de les utiliser au mieux.

10.1 LA BALLE QUI REBONDIT

Nous allons faire se déplacer une balle du bord gauche au bord droit de l'écran. La balle sera représentée par le petit carré dessiné par l'instruction `LINE(X,Y)-(X+3,Y+3),C,BF`. La méthode pour susciter l'illusion du mouvement consiste à dessiner la balle (d'une couleur différente de celle du fond) à une position X, l'effacer en la dessinant avec la couleur du fond et répéter le processus à X+1, X+2, etc. Le programme 10.1 crée cet effet.

```

0  "" 10-1 : Ball ""
10 SCREEN 1:CLS
20 FOR X=0 TO 315
30   C=3:GOSUB 1000
40   C=0:GOSUB 1000
50 NEXT
60 GOTO 20
1000 ' Draw the ball
1010 LINE(X,100)-(X+3,103),C,BF
1020 RETURN

```

PROGRAMME 10.1

Bien que le programme 10.1 soit un des exemples d'animation les plus élémentaires, il en illustre les quatre étapes obligatoires.

1. Dessiner l'objet.
2. Calculer la position suivante.
3. Effacer l'objet précédent, dessiner à la nouvelle position.
4. Aller à l'étape 2.

Nous allons maintenant faire rebondir la balle d'un mur à l'autre. Pour que la balle se déplace vers la droite, le programme 10.2 ajoute 1 (le contenu de D) à la coordonnée horizontale de la balle. Quand X devient supérieur à 316, la ligne 30 soustrait D de X pour ramener la balle à l'intérieur de l'écran, et change le signe de D. Maintenant que D est négatif, si on l'ajoute à X (ligne 60) on décrémente en fait X de 1, et la balle se déplace vers la gauche. Quand X devient inférieur à 0, le processus est répété et la balle commence à se déplacer de nouveau vers la droite. Le processus se répète ainsi indéfiniment.

```

0  "" 10-2 : Bouncing ball ""
10 SCREEN 1:CLS
20 D=1
30 IF X<0 OR X>316
   THEN
     X=X-D:D=-D:GOTO 30
40 LINE(PX,100)-(PX+3,103),0,BF
50 LINE(X,100)-(X+3,103),3,BF
60 PX=X:X=X+D:GOTO 30

```

PROGRAMME 10.2

Comme la valeur de X est changée à la ligne 60, on utilise PX pour mémoriser temporairement la coordonnée horizontale de la dernière balle dessinée afin de pouvoir l'effacer facilement. On peut obtenir le

même effet sans utiliser PX (comme on l'a fait au programme 10.1) en ajoutant D à X entre l'étape qui efface la balle et celle qui la dessine. Toutefois le laps de temps entre l'effacement et le dessin doit être minimisé ou l'image clignotera tout le temps et le résultat sera ennuyeux. En utilisant PX il est également possible de changer la vitesse de la balle en changeant simplement D. Essayez le programme 10.2 avec des valeurs différentes de 1.

10.2 L'ANIMATION AVEC DRAW

Dans ce paragraphe nous allons créer un robot animé en utilisant l'instruction DRAW. Le mouvement ira de la gauche vers la droite. La figure 10.1 montre, agrandis, les deux robots. L'animation pourrait être réalisée avec un seul robot mais le mouvement aurait l'air trop raide. Le sous-programme 500 du programme 10.3 dessine le robot #1 et le sous-programme 600 dessine le robot #2. L'instruction PRESET est utilisée pour placer le curseur (LRP) au point (X,100) ; la couleur 3 est utilisée pour tracer et la couleur 0 pour effacer :

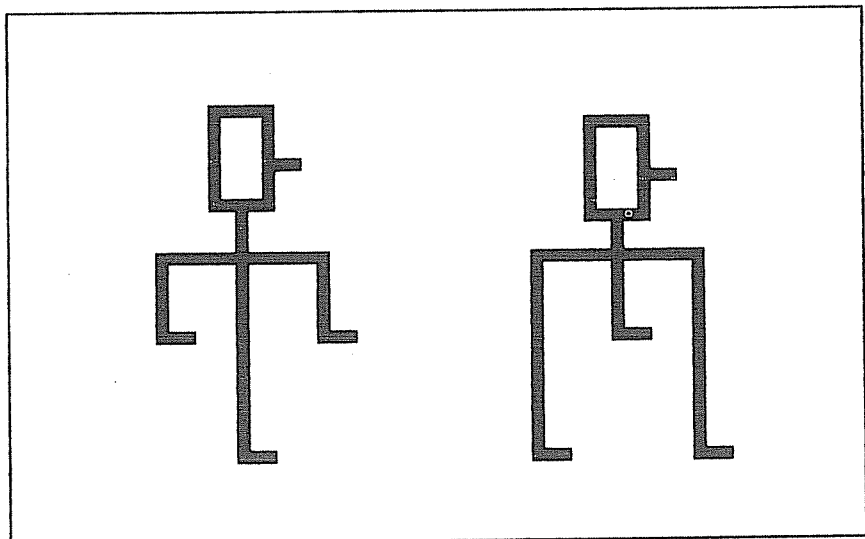


Fig. 10.1

```
0 '** 10-3 : Animation with DRAW **
5 DEFINT A-Z
10 SCREEN 1:CLS:X=3
20 COL=3:GOSUB 500
30 GOSUB 1000
40 COL=0:GOSUB 500
50 GOSUB 1000
```

```

60 X=X+2
70 COL=3:GOSUB 600
80 GOSUB 1000
90 COL=0:GOSUB 600
100 GOSUB 1000
110 X=X+2:GOTO 20
500 GOSUB 1000
510 PRESET(X,100)
520 DRAW"C=COL; R4 D4 R2 L2 D3 L4 U6"
530 DRAW"BM+2,+6 D4 L6 D6 R2"
540 DRAW"BM+4,-6 D15 R2 BM-2,-15 R6 D6 R2"
550 RETURN
600 'robot right 2
610 PRESET(X,100)
620 DRAW"C=COL; BM+0,1 R4 D4 R2 L2 D3 L4 U6"
630 DRAW"BM+2,+6 D3 L6 D15 R2"
640 DRAW"BM+4,-15 D6 R2 BM-2,-6 R6 D15 R2"
650 RETURN
1000 'Delay
1005 FOR I=1 TO 120
1010 NEXT
1020 RETURN

```

PROGRAMME 10.3

A cause du temps nécessaire pour tracer et effacer chaque robot, le mouvement n'est pas très bon. Des formes plus grandes et plus complexes rendront le clignotement pire encore. DRAW ne peut être utilisé que pour l'animation d'objets simples et petits.

10.3 L'ANIMATION DE FIGURES COMPLEXES

Les instructions GET et PUT sont spécialement appropriées pour l'animation. Comme les formes peuvent être dessinées à n'importe quel rythme avant d'être copiées dans les tableaux correspondants, on peut animer des figures aussi complexes soient-elles avec cette technique. Nous allons décrire ici la manière la plus simple de réaliser l'animation avec GET et PUT, c'est-à-dire l'animation sans fond et avec un mouvement restreint. En supposant qu'il n'y ait qu'une forme engagée dans l'animation, les étapes de base sont les suivantes :

1. Dessiner la forme et la ranger (GET) dans un tableau.
2. Effacer l'écran.
3. Ecrire (PUT) la forme sur l'écran sous le mode PSET.
4. Calculer les nouvelles coordonnées de la forme.
5. Aller à l'étape 3.

Notez que la forme n'est pas effacée avant d'être dessinée à sa nouvelle position. Quand une figure est écrite (PUT) sur l'écran sous le mode PSET, chaque pixel à l'intérieur de la zone du rectangle copié à l'origine est

effacé. Il est donc important de ranger (GET) une zone assez large pour effacer la forme précédente.

Dans le premier exemple d'animation avec GET et PUT, le programme 10.4, nous utiliserons les formes du programme 10.3. Les lignes 20 à 80 rangent (GET) les deux formes dans les tableaux ROBOT1 et ROBOT2. Aux lignes 100 à 170 les robots sont remis (PUT) sur l'écran en utilisant la variable GUIDE pour alterner les formes. La figure 10.2 montre la zone couverte par chaque rectangle. Le robot précédent (en pointillé) est complètement recouvert par le nouveau.

```

0 '* 10-4 : Animation with GET & PUT *
5 DEFINT A-Z
6 N=300: DIM ROBOT1(N), ROBOT2(N)
10 SCREEN 1: CLS: X=10
20 COL=3: GOSUB 500
30 GET (3,100)-(21,126), ROBOT1
60 CLS: X=10
70 COL=3: GOSUB 600
80 GET (3,100)-(21,126), ROBOT2
90 CLS
100 X=0: GUIDE=1
110 WHILE X<302
120   ON GUIDE GOTO 130,140
130   PUT (X,90), ROBOT1, PSET: GOTO 150
140   PUT (X,90), ROBOT2, PSET
150   X=X+2
155 GOSUB 1000
160   GUIDE=GUIDE+1:
      IF GUIDE=3
      THEN
        GUIDE=1
170 WEND
180 END
500 GOSUB 1000
510 PRESET (X,100)
520 DRAW"C=COL; R4 D4 R2 L2 D3 L4 U6"
530 DRAW"BM+2,+6 D4 L6 D6 R2"
540 DRAW"BM+4,-6 D15 R2 BM-2,-15 R6 D6 R2"
550 RETURN
600 'robot right 2
610 PRESET (X,100)
620 DRAW"C=COL; BM+0,1 R4 D4 R2 L2 D3 L4 U6"
630 DRAW"BM+2,+6 D3 L6 D15 R2"
640 DRAW"BM+4,-15 D6 R2 BM-2,-6 R6 D15 R2"
650 RETURN
1000 'Delay
1005 FOR I=1 TO 300
1010 NEXT
1020 RETURN

```

PROGRAMME 10.4

Chaque robot est dessiné deux pixels à droite du précédent. Si à la ligne 150, X est incrémenté d'une valeur supérieure à 2, les robots se déplaceront plus rapidement, mais ils ne seront effacés complètement que

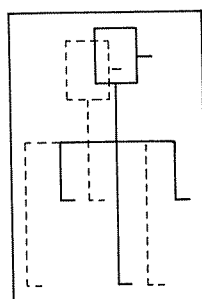


Fig. 10.2

si la zone originellement mémorisée est assez grande pour couvrir la forme précédente. Avec la zone utilisée aux lignes 30 et 80, on peut utiliser en toute sécurité un incrément allant jusqu'à 3, de plus grands laisseraient des parties des formes sur l'écran. La figure 10.3 montre la trace laissée quand on utilise un incrément de 5.

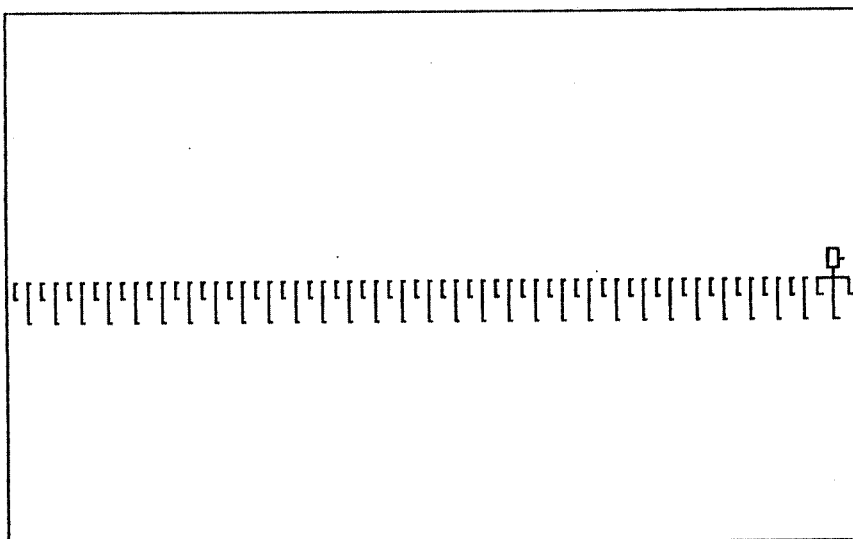


Fig. 10.3

Dans le second exemple, le programme 10.5, nous utiliserons une soucoupe volante en train de tourner pour montrer l'animation avec GET et PUT. Le sous-programme 300 dessine le corps de la soucoupe, et les lignes 40 à 170 tracent les parties qui différencient chacune des trois formes, avec des fenêtres à différentes positions et une paire de jambes différente pour chacune d'elles. La figure 10.4 montre les trois formes agrandies.

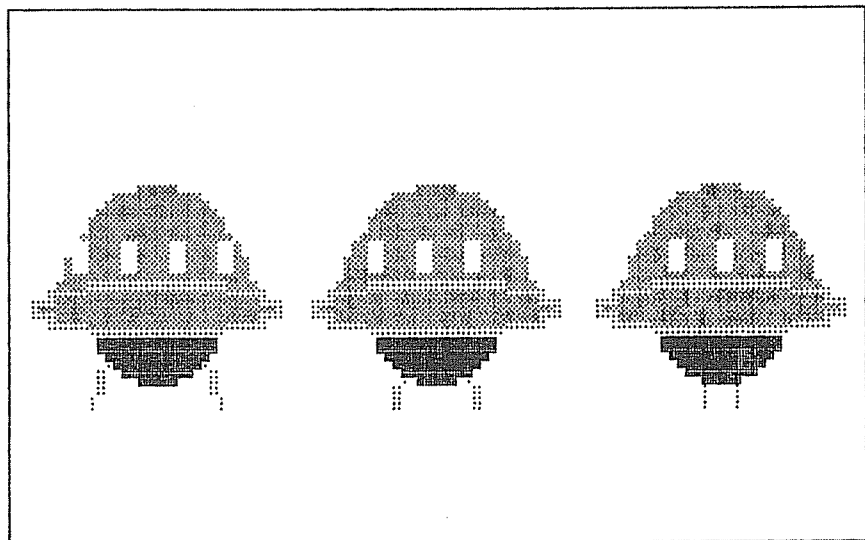


Fig. 10.4

Quand chaque forme est copiée dans un tableau, on utilise un rectangle de deux pixels plus large que le minimum nécessaire dans chaque direction afin d'effacer avec la soucoupe précédente. La trajectoire de la soucoupe est différente de celle des exemples précédents. Elle est semblable à la balle qui rebondit parce qu'elle va de gauche à droite, mais elle monte et elle descend, suivant le chemin d'un V. On prend D comme incrément pour X (le déplacement horizontal) et B pour Y (déplacement vertical). Le son produit par l'instruction de la ligne 250 ajoute quelque intérêt au mouvement.

```

0 '** 10-5 : Animated flying saucer **
10 DEFINT A-Z
20 N=145: DIM SAUCER1(N), SAUCER2(N), SAUCER3(N)
30 SCREEN 1: COLOR 0,0: CLS: X=10
40 LINE (42,62)-(50,40), 1
50 LINE (58,62)-(50,40), 1
60 GOSUB 300:
  FOR I=40 TO 60 STEP 6:
    LINE (I,42)-(I+1,45), 0,BF:
  NEXT
70 GET (33,33)-(67,64), SAUCER1
80 CLS
90 LINE (45,62)-(50,40), 1
100 LINE (55,62)-(50,40), 1
110 GOSUB 300:
  FOR I=42 TO 55 STEP 6:
    LINE (I,42)-(I+1,45), 0,BF:
  NEXT
120 GET (33,33)-(67,64), SAUCER2
130 CLS
140 LINE (48,62)-(50,40), 1

```



```

150 LINE(52,62)-(50,40),1
160 GOSUB 300:
    FOR I=44 TO 57 STEP 6:
        LINE(I,42)-(I+1,45),0,BF:
    NEXT
170 GET(33,33)-(67,64),SAUCER3
180 CLS
190 X=90:Y=100:GUIDE=1:D=1:B=1
200 IF X<0
    THEN
        X=0:D=1:B=1:GOTO 200
205 IF X>285
    THEN
        X=285:D=-1:B=1:GOTO 205
207 IF Y>168
    THEN
        Y=168:B=-B:GOTO 207
210 ON GUIDE GOTO 220,230,240
220 PUT(X,Y),SAUCER1:PSET:GOTO 250
230 PUT(X,Y),SAUCER2,PSET:GOTO 250
240 PUT(X,Y),SAUCER3,PSET
250 GOSUB 390
260 X=X+D:Y=Y+B
270 GUIDE=GUIDE+1:
    IF GUIDE=4
        THEN
            GUIDE=1
280 GOTO 200
290 END
300 'Draw basic saucer
310 CIRCLE(50,51),8,3,,,1
320 PAINT(50,56),3,3
330 CIRCLE(50,50),15,2,,,2
340 PAINT(50,50),2,2
350 CIRCLE(50,50),15,2,0,3.14,1.3
360 PAINT(50,45),2,2
370 CIRCLE(50,50),15,1,,,2
380 RETURN
390 SOUND 37+20*RND,.1:RETURN

```

PROGRAMME 10.5

Dans le troisième et dernier exemple de ce paragraphe, nous reproduirons le mouvement d'un homme qui marche. La figure 10.5 montre, agrandies, les trois figures utilisées. La première est produite par les lignes 20 à 74 du programme 10.6 et copiée dans le tableau MAN1. La deuxième est tracée par les lignes 100 à 164 et copiée dans MAN2 et la troisième est produite par les lignes 200 à 264 et copiée dans MAN3.

L'animation effective des lignes 500 à 590 diffère des deux exemples précédents en ce que la forme est tracée trois fois : par exemple la ligne 530 écrit (PUT) le tableau MAN1 à (X,100), MAN2 à (X+20,100) et MAN3 à (X+35),100). Ceci produit les trois silhouettes, chacune dans une phase différente de la marche.

La même méthode que l'on vient d'expliquer est utilisée au paragraphe 11.2.9 pour montrer l'animation d'un objet à trois dimensions tournant dans l'espace.

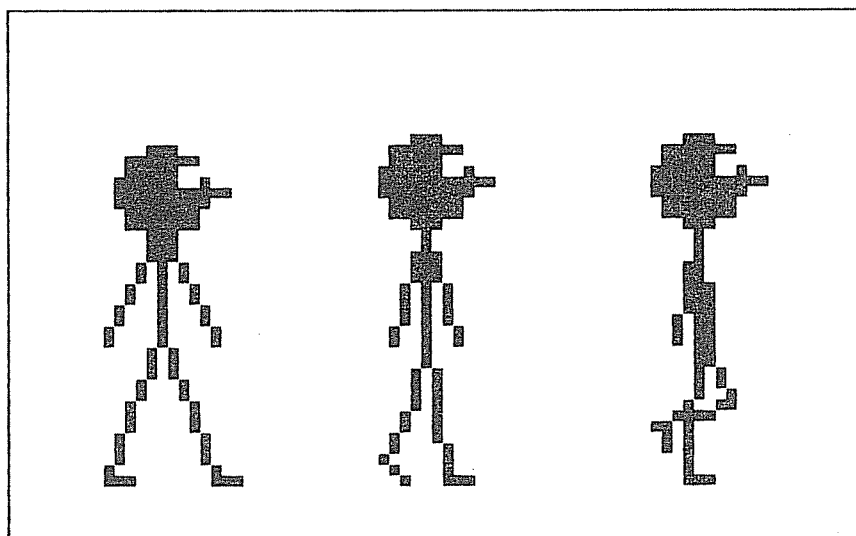


Fig. 10.5

```

0 '** 10-6 : Animated men **
5 N=34: DIM MAN1(N),MAN2(N),MAN3(N)
10 SCREEN 1:CLS
20 CIRCLE(6,6),4,3,,,1
22 PAINT(6,6),3,3
24 LINE(8,4)-(9,5),0,BF
26 LINE(10,6)-(12,6)
30 LINE(6,9)-(6,19)
40 LINE-(11,33)
50 LINE-(13,33)
60 LINE(6,19)-(1,33)
70 LINE-(3,33)
72 LINE(6,10)-(11,20)
74 LINE(6,10)-(1,20)
80 GET(0,1)-(13,33),MAN1
90 CLS
100 CIRCLE(6,5),4,3,,,1
102 PAINT(6,5),3,3
104 LINE(8,3)-(9,4),0,BF
106 LINE(10,5)-(12,5)
110 LINE(6,9)-(6,19)
120 LINE-(8,33)
130 LINE-(10,33)
140 LINE(6,19)-(5,26)
150 LINE-(2,31)
160 LINE-(4,33)
162 LINE(6,10)-(9,20)
164 LINE(6,10)-(3,20)
170 GET(0,1)-(13,33),MAN2
180 CLS
200 CIRCLE(6,5),4,3,,,1
202 PAINT(6,5),3,3
204 LINE(8,3)-(9,4),0,BF
206 LINE(10,5)-(12,5)

```

```

210 LINE(6,9)-(6,19)
220 LINE-(5,33)
230 LINE-(7,33)
240 LINE(6,19)-(9,26)
250 LINE-(2,28)
260 LINE-(3,30)
262 LINE(6,10)-(7,20)
264 LINE(6,10)-(4,20)
270 GET(0,1)-(13,33),MAN3
280 CLS
500 X=0:I=1
510 WHILE X<272
520   ON I GOTO 530,540,550
530   PUT(X,100),MAN1,PSET:
      PUT(X+20,100),MAN2,PSET:
      PUT(X+35,100),MAN3,PSET:GOTO 540
540   PUT(X,100),MAN2,PSET:
      PUT(X+20,100),MAN3,PSET:
      PUT(X+35,100),MAN1,PSET:GOTO 540
550   PUT(X,100),MAN3,PSET:
      PUT(X+20,100),MAN1,PSET:
      PUT(X+35,100),MAN2,PSET
560   I=I+1:
      IF I=4
      THEN
        I=1
570   X=X+1
580 FOR J=1 TO 90:
      NEXT
590 WEND
600 LOCATE 15

```

PROGRAMME 10.6

10.4 L'ANIMATION AVEC UN FOND COMPLEXE

Des fonds simples permettent l'utilisation de blocs pleins pour effacer la forme en déplacement. Malheureusement, la forme doit se déplacer presque toujours sur des fonds non vides. Dans ces cas, un bloc plein effacera non seulement la figure en mouvement, mais aussi le fond. Pour vous en convaincre, essayez la combinaison des programmes 10.7 et 10.5 :

```

182 FOR I=0 TO 319 STEP 5
183   LINE(I,0)-(I,199)
184 NEXT

```

PROGRAMME 10.7

Nous allons voir deux manières de résoudre ce problème : la première utilise le mode XOR de l'instruction PUT et la seconde mémorise dans un tableau la partie de l'écran qui sera affectée par la forme, avant qu'elle ne le soit effectivement.

10.4.1 L'animation avec XOR

Nous rappelons (voir paragraphe 9.12) que lorsqu'une figure est tracée sur l'écran avec l'instruction PUT sous le mode XOR, la fonction XOR est exécutée entre chaque pixel de l'écran et son correspondant sur la figure. Si on réexécute la fonction XOR une deuxième fois, au même endroit, l'écran est restauré. Ceci rend ce mode idéal pour l'animation sur des fonds complexes parce que l'objet peut se déplacer facilement sans changer quoi que ce soit sur son chemin.

Les étapes pour ce genre d'animation sont légèrement différentes de celles expliquées plus haut. Supposez qu'il y ait plusieurs formes (comme les trois utilisées pour l'homme en train de marcher au programme 10.6), chacune est identifiée avec un numéro de forme, les étapes à observer étant les suivantes :

1. Dessiner et ranger (GET) les figures dans des tableaux. Effacer l'écran.
2. Ecrire (PUT) la première forme sur l'écran sous le mode (par défaut) XOR. Mémoriser ses coordonnées et son numéro de forme.
3. Calculer la nouvelle forme et ses coordonnées.
4. Ecrire (PUT) la forme dont le numéro a été mémorisé à l'étape 2 au point où elle était précédemment tracée (ceci l'efface).
5. Ecrire (PUT) la forme suivante à sa nouvelle position. Mémoriser son numéro de forme et ses coordonnées.
6. Aller à l'étape 3.

Dans le premier exemple de ce type d'animation, le programme 10.8, les lignes 182 à 187 dessinent une grille qui servira de fond pour la soucoupe volante du programme 10.5. A la ligne 190, X (position horizontale), Y (position verticale), PX (X de la forme précédente) et PY (Y de la forme précédente) sont initialisés. On affecte à D et B (les incréments horizontal et vertical) la valeur 1. PG (la soucoupe précédente, ou la dernière soucoupe dessinée) a le numéro 3, et la soucoupe # 3 est écrite (PUT) sur l'écran en (PX,PY). C'est l'équivalent de l'étape 2 vue plus haut. GUIDE (la soucoupe en cours) a le numéro 1. Les instructions IF des lignes 200 à 207 sont sautées la première fois, et aux lignes 210 à 240 la soucoupe précédente (numéro 3 dans ce cas) est écrite (PUT) une seconde fois, rétablissant ainsi le fond. C'est l'équivalent de l'étape 4. Aux lignes 250 à 256 la soucoupe suivante est écrite (PUT) sur l'écran (étape 5), les anciennes coordonnées ainsi que le numéro de GUIDE sont sauvegardés, la nouvelle soucoupe et ses coordonnées sont calculées aux lignes 265 à 270 et la boucle est répétée.

```
0 *** 10-B : Animation with XOR **
10 DEFINT A-Z
20 N=145: DIM SAUCER1(N), SAUCER2(N), SAUCER3(N)
```

```

30 SCREEN 1:COLOR 0,0:CLS:X=10
40 LINE(42,62)-(50,40),1
50 LINE(58,62)-(50,40),1
60 GOSUB 300:
  FOR I=40 TO 60 STEP 6:
    LINE(I,42)-(I+1,45),0,BF:
  NEXT
70 GET(33,33)-(67,64),SAUCER1
80 CLS
90 LINE(45,62)-(50,40),1
100 LINE(55,62)-(50,40),1
110 GOSUB 300:
  FOR I=42 TO 55 STEP 6:
    LINE(I,42)-(I+1,45),0,BF:
  NEXT
120 GET(33,33)-(67,64),SAUCER2
130 CLS
140 LINE(48,62)-(50,40),1
150 LINE(52,62)-(50,40),1
160 GOSUB 300:
  FOR I=44 TO 57 STEP 6:
    LINE(I,42)-(I+1,45),0,BF:
  NEXT
170 GET(33,33)-(67,64),SAUCER3
180 CLS
182 FOR I=0 TO 319 STEP 10
183   LINE(I,0)-(I,199),1
184 NEXT
185 FOR I=0 TO 199 STEP 10
186   LINE(0,I)-(319,I),1
187 NEXT
190 X=90:Y=100:GUIDE=1:PG=3:PX=X:PY=Y:
  PUT(X,Y),SAUCER3:D=1:B=1
200 IF X<0
  THEN
    X=0:D=1:B=1:GOTO 200
205 IF X>285
  THEN
    X=285:D=-1:B=1:GOTO 205
207 IF Y>168
  THEN
    Y=168:B=-B:GOTO 207
210 ON PG GOTO 220,230,240
220   PUT(PX,PY),SAUCER1:GOTO 250
230   PUT(PX,PY),SAUCER2:GOTO 250
240   PUT(PX,PY),SAUCER3
250 ON GUIDE GOTO 252,254,256
252   PUT(X,Y),SAUCER1:GOTO 260
254   PUT(X,Y),SAUCER2:GOTO 260
256   PUT(X,Y),SAUCER3
260 SOUND 37+20*RND,.1
265 PX=X:PY=Y:PG=GUIDE:X=X+D:Y=Y+B
270 GUIDE=GUIDE+1:
  IF GUIDE=4
  THEN
    GUIDE=1
280 GOTO 200
290 END
300 'Draw basic saucer
310 CIRCLE(50,51),8,3,,,1

```

```

320 PAINT(50,56),3,3
330 CIRCLE(50,50),15,2,,.2
340 PAINT(50,50),2,2
350 CIRCLE(50,50),15,2,0,3.14,1.3
360 PAINT(50,45),2,2
370 CIRCLE(50,50),15,1,,.2
380 RETURN

```

PROGRAMME 10.8

Une des caractéristiques du mode XOR est que les couleurs de l'objet original ne sont pas préservées sur l'écran, mais dépendent du fond. C'est ceci qui rend possible la restauration quand le tableau est écrit (PUT) une deuxième fois. Les parties du fond ayant la couleur 0 préservent les couleurs de l'objet sur lequel on a appliqué la fonction XOR. A part la couleur 0, le fond du programme 10.8 avait seulement des lignes à la couleur 1, et les changements de la soucoupe volante ne sont pas très discernables. Ils le seront dans l'exemple suivant.

Une autre caractéristique de ce type d'animation est que l'objet se déplaçant est constamment en train de clignoter. Comme l'objet est complètement effacé (et le fond restauré à son état original), il y a toujours un laps de temps pendant lequel l'objet n'est pas présent sur l'écran. Dans l'animation du paragraphe 10.3, ce clignotement n'existait pas parce que l'objet était effacé et redessiné simultanément. Quand l'objet est petit et quand le laps de temps entre l'effacement et le PUT suivant est minimisé, cet effet « clignotant » est difficilement visible. Avec de grands objets, cela peut devenir ennuyeux. Au paragraphe 10.6 nous montrerons une méthode pour circonvenir ce problème, mais qui aura malheureusement aussi quelques limites.

Le second exemple d'animation avec le mode XOR de PUT fera se déplacer la figure au travers de bandes de toutes les couleurs. Au programme 10.9 ces bandes sont créées aux lignes 30 à 40. Le mouvement de la figure est contrôlé par les quatre touches de déplacement du curseur et, à la différence des programmes précédents, ici les touches à deux codes sont utilisées (et non les nombres). La vitesse du mouvement peut être changée en appuyant sur le « A » (augmentée) ou le « Z » (diminuée).

```

0 ' ** 10-9 : Animation of flying **
5 '* saucer with complex background *
10 SCREEN 1:COLOR 1,2:CLS:S=1
20 CIRCLE (6,6),5:PAINT (6,6),2,3
24 PSET (5,0),1:DRAW"D12 R2 U12 L2":
   PAINT (6,6),1,1
26 DIM M(100):GET (0,0)-(11,12),M:CLS
30 FOR I=0 TO 299 STEP 20
32   LINE (I,0)-(I+20,199),J,BF
34   J=J+1:IF J=4 THEN J=0
40 NEXT:PUT (0,0),M
50 W$=INKEY$:
   IF LEN(W$)=2
   THEN

```

```

        W=ASC(RIGHT$(W$,1)):GOTO 200
55 IF LEN(W$)=2
    THEN
        W=ASC(RIGHT$(W$,1)):GOTO 200
60 IF W$<>"A"AND W$<>"a"
    THEN
        80
70 IF S<10
    THEN
        S=S+.5:GOTO 50
    ELSE
        50
80 IF W$<>"Z"AND W$<>"z"
    THEN
        IF W$=""
            THEN
                1000
            ELSE
                SOUND 40,5:GOTO 50
90 IF S>0
    THEN
        S=S-.5
100 GOTO 50
200 IF W=77
    THEN
        G=1:GOTO 1000
210 IF W=75
    THEN
        G=2:GOTO 1000
220 IF W=72
    THEN
        G=3:GOTO 1000
230 IF W=80
    THEN
        G=4:GOTO 1000
240 GOTO 1000
1000 ON G GOTO 1010,1020,1030,1040
1010 IF X+S<308
    THEN
        X=X+S:GOTO 2000
    ELSE
        2000
1020 IF X-S>=0
    THEN
        X=X-S:GOTO 2000
    ELSE
        2000
1030 IF Y-S>0
    THEN
        Y=Y-S:GOTO 2000
    ELSE
        2000
1040 IF Y+S<187
    THEN
        Y=Y+S:GOTO 2000
    ELSE
        2000
2000 PUT (PX,PY),M:PUT (X,Y),M:
    PX=X:PY=Y:GOTO 50

```

PROGRAMME 10.9

Le troisième exemple utilisera les silhouettes de l'homme en train de marcher du programme 10.6. Les lignes 400 à 470 du programme 10.10 dessinent la maison et l'horizon que l'on voit sur la figure 10.6.

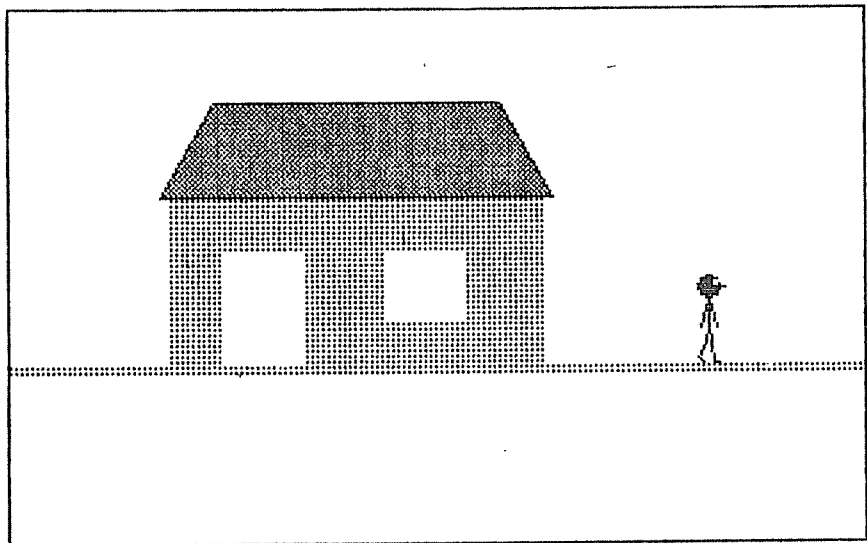


Fig. 10.6

Les lignes 500 à 590 réalisent l'animation. Notez qu'à la différence du programme 10.6, on peut utiliser ici n'importe quel incrément, parce que l'effacement de la figure précédente n'a rien à voir avec la taille du rectangle sauvegardé au départ par GET. En changeant l'incrément à la ligne 570, l'homme marchera plus vite ou plus lentement. Notez les changements de couleur lorsque le mobile passe devant le plan de la maison.

```
0 ' ** 10-10 : Man moving over house **
5 N=34: DIM MAN1(N),MAN2(N),MAN3(N)
10 SCREEN 1: COLOR 1,0:CLS
20 CIRCLE(6,6),4,3,,,1
22 PAINT(6,6),3,3
24 LINE(8,4)-(9,5),0,BF
26 LINE(10,6)-(12,6)
30 LINE(6,9)-(6,19)
40 LINE-(11,33)
50 LINE-(13,33)
60 LINE(6,19)-(1,33)
70 LINE-(3,33)
72 LINE(6,10)-(11,20)
74 LINE(6,10)-(1,20)
80 GET(0,1)-(13,33),MAN1
90 CLS
100 CIRCLE(6,5),4,3,,,1
102 PAINT(6,5),3,3
```



```

104 LINE(8,3)-(9,4),0,BF
106 LINE(10,5)-(12,5)
110 LINE(6,9)-(6,19)
120 LINE-(8,33)
130 LINE-(10,33)
140 LINE(6,19)-(5,26)
150 LINE-(2,31)
160 LINE-(4,33)
162 LINE(6,10)-(9,20)
164 LINE(6,10)-(3,20)
170 GET(0,1)-(13,33),MAN2
180 CLS
200 CIRCLE(6,5),4,3,,,1
202 PAINT(6,5),3,3
204 LINE(8,3)-(9,4),0,BF
206 LINE(10,5)-(12,5)
210 LINE(6,9)-(6,19)
220 LINE-(5,33)
230 LINE-(7,33)
240 LINE(6,19)-(9,26)
250 LINE-(2,28)
260 LINE-(3,30)
262 LINE(6,10)-(7,20)
264 LINE(6,10)-(4,20)
270 GET(0,1)-(13,33),MAN3
280 CLS
400 LINE(60,70)-(200,133),1,BF
410 LINE(80,90)-(110,133),0,BF
415 LINE(140,90)-(170,115),0,BF
420 LINE(57,70)-(203,70)
430 LINE(57,70)-(77,35)
440 LINE-(183,35)
450 LINE-(203,70)
460 PAINT(70,50),2,3
470 LINE(0,133)-(319,135),1,BF
500 X=0:I=1:PUT(0,100),MAN3
510 WHILE X<309
520   ON I GOTO 530,540,550
530   PUT(PX,100),MAN3:PUT(X,100),MAN1:
      GOTO 560
540   PUT(PX,100),MAN1:PUT(X,100),MAN2:
      GOTO 560
550   PUT(PX,100),MAN2:PUT(X,100),MAN3
560   I=I+1:
      IF I=4
      THEN
        I=1
570   PX=X:X=X+3
580   FOR J=1 TO 90:
      NEXT
590 WEND

```

PROGRAMME 10.10

10.4.2 Copie et restauration

Comme nous l'avons expliqué au paragraphe précédent, lorsqu'on exécute la fonction XOR entre une forme et l'écran, les couleurs qui sont

en fait affichées (PUT) sur l'écran ne correspondent pas nécessairement à celles d'origine. S'il est impératif de conserver ces couleurs, et que la forme se déplace sur un fond complexe, la solution consiste à copier la zone destinée à être traversée avant que le mobile ne soit affiché (PUT) sur l'écran sous le mode PSET (qui garantit que les couleurs d'origine sont préservées). Pour changer la position de l'objet, la partie de l'écran qui était sauvegardée précédemment est restituée et l'objet affiché (PUT) à son nouvel emplacement. Le programme 10.11 utilise le fond et l'objet du programme 10.9 pour faire la démonstration de cette technique :

```

0 ' ** 10-11 : Copy and restore **
10 SCREEN 1:COLOR 1,2:CLS:S=1
20 CIRCLE (6,6),5:PAINT (6,6),2,3
24 PSET (5,0),1:DRAW"D12 R2 U12 L2":
    PAINT (6,6),1,1
26 DIM M(100),T(100):GET (0,0)-(11,12),M:CLS
30 FOR I=0 TO 299 STEP 20
32   LINE (I,0)-(I+20,199),J,BF
34   J=J+1:
      IF J=4
      THEN
        J=0
40 NEXT
45 GET (0,0)-(11,12),T:PUT (0,0),M,PSET
47 PX=0:PY=0
50 W$=INKEY$:
    IF LEN(W$)=2
    THEN
      W=ASC(RIGHT$(W$,1)):GOTO 200
55 IF LEN(W$)=2
    THEN
      W=ASC(RIGHT$(W$,1)):GOTO 200
60 IF W$<>"A"AND W$<>"a"
    THEN
      80
70 IF S<10
    THEN
      S=S+.5:GOTO 50
    ELSE
      50
80 IF W$<>"Z"AND W$<>"z"
    THEN
      IF W$=""
      THEN
        1000
      ELSE
        SOUND 40,5:GOTO 50
90 IF S>0
    THEN
      S=S-.5
100 GOTO 50
200 IF W=77
    THEN
      G=1:GOTO 1000
210 IF W=75
    THEN
      G=2:GOTO 1000
220 IF W=72

```

```

      THEN
        G=3:GOTO 1000
230 IF W=80
      THEN
        G=4:GOTO 1000
240 GOTO 1000
1000 ON G GOTO 1010,1020,1030,1040
1010 IF X+S<308
      THEN
        X=X+S:GOTO 2000
      ELSE
        2000
1020 IF X-S>=0
      THEN
        X=X-S:GOTO 2000
      ELSE
        2000
1030 IF Y-S>0
      THEN
        Y=Y-S:GOTO 2000
      ELSE
        2000
1040 IF Y+S<187
      THEN
        Y=Y+S:GOTO 2000
      ELSE
        2000
2000 PUT (PX,PY),T,PSET:
      GET(X,Y)-(X+11,Y+12),T:PUT (X,Y),M,PSET:
      PX=X:PY=Y:GOTO 50

```

PROGRAMME 10.11

Cette méthode pose cependant plusieurs problèmes : elle augmente le scintillement parce qu'il faut ajouter le temps pour copier le fond au temps nécessaire à la restitution. Pour cette même raison, la méthode est lente. Le mode PSET non seulement dessine l'objet avec ses couleurs d'origine, mais aussi dessine la partie du fond couverte par le rectangle utilisé dans GET. Dans le programme 10.11 l'objet est rond et il y a des parties du rectangle qui ne sont évidemment pas couvertes par lui (comme ce sera le cas la plupart du temps). Ces parties sont effacées du fond chaque fois que l'objet est écrit (PUT) sur l'écran, créant alors une image confuse. Quand l'objet est rectangulaire (et que le rectangle a ses côtés parallèles aux axes des X et des Y) cette méthode peut donner une animation correcte. Si on ajoute le programme 10.12 au programme 10.11 la forme sera rectangulaire.

```
15 LINE(0,0)-(11,12),2,B:PAINT(5,5),1,2
```

PROGRAMME 10.12

10.5 MOUVEMENT SANS SCINTILLEMENT AVEC UN FOND COMPLEXE

Quand une forme est animée avec la méthode XOR et qu'elle est amenée à un emplacement qui chevauche sa position précédente, le court laps de temps pendant lequel la forme n'est pas sur l'écran provoque un scintillement qui peut être très gênant, surtout si la forme est grande. Nous allons présenter ici une méthode pour produire une animation sans scintillement avec un fond complexe. Le processus est légèrement plus compliqué que les précédents, aussi nous commencerons par un exemple. Nous allons dessiner une roue qui se déplacera sur l'écran de gauche à droite. La figure 10.7 montre les deux roues qui seront utilisées.

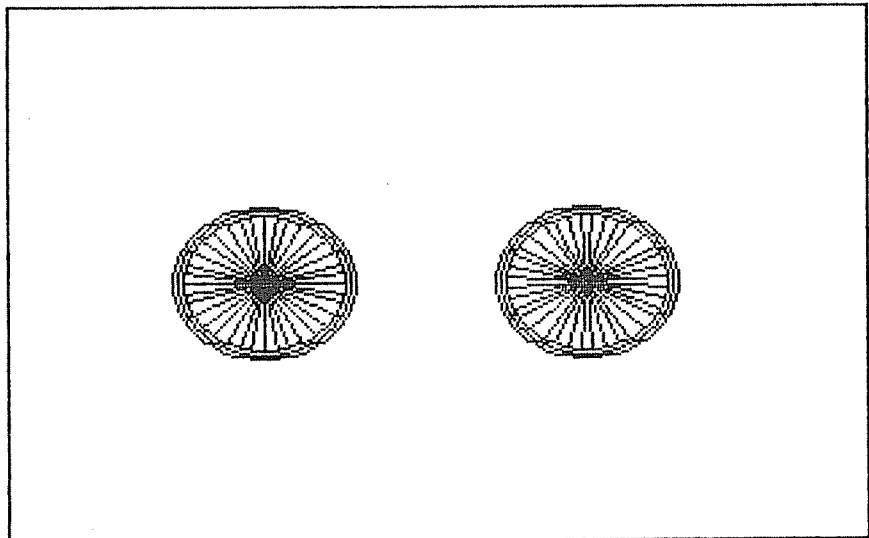


Fig. 10.7

Le déplacement des roues sera de deux pixels sur la droite. Nous commençons par dessiner et ranger (GET) les deux roues dans les tableaux WHEEL1 et WHEEL2. Nous effaçons l'écran. Nous écrivons (PUT) WHEEL1 à (0,0) et faisons XOR sur WHEEL2 deux pixels à droite avec les instructions : PUT(0,0),WHEEL1:PUT(2,0),WHEEL2,XOR. Nous rangeons (GET) alors le dessin résultant (qui à ce moment est une forme confuse) dans le tableau A1, qui est deux fois plus grand que WHEEL1 ou WHEEL2. Nous effaçons l'écran et répétons l'opération en changeant l'ordre, c'est-à-dire en écrivant (PUT)WHEEL2 d'abord et WHEEL1 ensuite, avec les instructions PUT(0,0),WHEEL2:PUT(2,0),WHEEL1, XOR. Nous rangeons (GET) ce nouveau dessin dans A2.

Nous commençons alors l'animation : la première roue est dessinée à la position (horizontale) 0. Si nous voulions effacer WHEEL1 et dessi-

ner WHEEL2 à la position 2 avec la méthode expliquée au paragraphe 10.4.1, nous devrions exécuter un XOR entre WHEEL1 et l'écran à la position 0 et faire la même chose avec WHEEL2 à la position 2. Ainsi A1 contient WHEEL1 « XORé » à l'emplacement 0 et WHEEL2 « XORé » à l'emplacement 2, les deux étapes en une seule. Si nous « XORons » A1 à l'emplacement 0, cela effacera à la fois WHEEL1 et dessinera WHEEL2 à l'emplacement 2. Maintenant, en répétant ce processus, nous écrivons A2 à l'emplacement 2 effaçant WHEEL2 et dessinant WHEEL1. Le programme 10.13 fait se déplacer les roues sur un fond composé de rubans colorés.

```

0  '** 10-13 : Flicker-free wheels **
20 F=1.745329E-02:
   DIM WHEEL1(1650),WHEEL2(1650),
       A1(2000),A2(2000)
30 SCREEN 1:COLOR 0,1:CLS:PS=0
40 FOR S=10 TO 360 STEP 10
50   CIRCLE(160,100),30,1,-PS*F,-S*F
60 NEXT
70 CIRCLE(160,100),32,2
80 CIRCLE(160,100),34,1
90 GET(125,72)-(194,128),WHEEL1
100 CLS:PS=5:CIRCLE(160,100),30,1,-355*F,-5*F
110 FOR S=15 TO 355 STEP 10
120   CIRCLE(160,100),30,1,-PS*F,-S*F
130 NEXT
140 CIRCLE(160,100),32,2
150 CIRCLE(160,100),34,1
160 GET(125,72)-(194,128),WHEEL2
170 CLS
180 PUT(0,0),WHEEL1:PUT(2,0),WHEEL2
190 GET(0,0)-(71,56),A1:CLS
200 PUT(0,0),WHEEL2:PUT(2,0),WHEEL1
210 GET(0,0)-(71,56),A2:CLS
212 FOR I=50 TO 250 STEP 10
213   LINE(I,31)-(I+10,86),COL,BF
214   COL=COL+1:
       IF COL=4
       THEN
         COL=0
215 NEXT
220 PUT(0,30),WHEEL1:X=0:GUIDE=1
230 WHILE X<250
240   ON GUIDE GOTO 250,260
250   PUT(X,30),A1:GUIDE=2:GOTO 270
260   PUT(X,30),A2:GUIDE=1
270   X=X+2
280 WEND

```

PROGRAMME 10.13

Revoyons le processus du programme 10.13 utilisant les tableaux. WHEEL1 était écrit (PUT) sur l'écran à l'emplacement 0. Au lieu de l'effacer avec un second PUT et de dessiner WHEEL2 à son nouvel emplacement deux pixels sur la droite, on a utilisé le tableau A1 parce qu'il

contenait déjà WHEEL1 « XORé » à l'emplacement 0 et WHEEL2 « XORé » à l'emplacement 2. Il pouvait sembler que quand deux roues étaient « XORées » l'une au-dessus de l'autre, la confusion qui en résultait ne serait d'aucune utilité ; toutefois le mode XOR préserve toujours l'information intacte. Il est possible, par exemple, de « XORer » un grand nombre de figures sur le même endroit et de les effacer une par une dans un ordre différent.

Le mouvement créé par cette méthode sans scintillement est supérieur à tous les précédents parce qu'il est très régulier et peut être utilisé sur des fonds complexes. Toutefois il a quelques inconvénients. Comme les tableaux qui contiendront les images « XORées » combinées doivent être créés avec le déplacement inclus, le mouvement des formes est limité à des emplacements spécifiques. Les tableaux combinés sont en général plus grands que les tableaux simples. Bien qu'on ait besoin que des tableaux combinés pendant l'animation on a besoin d'une image de départ (une des formes d'origine) pour commencer.

Nous allons présenter maintenant un second exemple d'animation sans scintillement, avec l'animation XOR afin de faire des comparaisons. Le programme 10.14 produit les deux chevaux que l'on voit à la figure 10.8 et les range (GET) dans les tableaux HORSE1 et HORSE2.

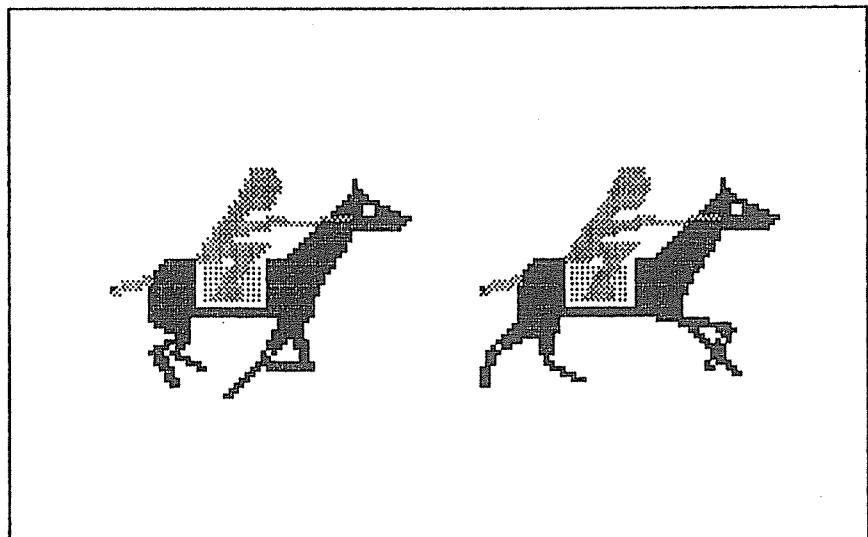


Fig. 10.8

Les figures sont combinées dans les tableaux B1 et B2. Dans la boucle FOR des lignes 479 à 490 les chevaux sont « XORés » à Y=100. Le mouvement du cheval du bas est beaucoup plus régulier que celui du haut.

```

0 '** 10-14 : Flicker-free horse **
5 DEFINT A-Z:
  DIM HORSE1(500),HORSE2(500),
    B1(500),B2(500)
10 SCREEN 1:COLOR 0,0:CLS
20 'draw horse's body
30 DRAW"bm10,20 e3 r22 e13 u2"
40 LINE -STEP(1,3):LINE -STEP(9,4)
50 LINE -STEP(-2,2):LINE -STEP(-8,0)
60 LINE -STEP(-8,13):LINE -STEP(0,3)
70 LINE -STEP(-30,0):LINE -STEP(0,-7)
80 PAINT(20,20),3,3
90 C=2:C1=1:GOSUB 2000
100 GET(3,0)-(58,27),B1
110 C=2:C1=0:GOSUB 2000
120 GET(3,0)-(58,27),B2
130 C=1:C1=2:GOSUB 2000
300 'draw feet 1
310 C=3
320 LINE(10,28)-(15,31),C:
  LINE-STEP(-5,3):LINE-STEP(4,6),C
330 LINE-STEP(1,0),C:LINE-STEP(-3,-6),C
340 LINE-STEP(-1,0),C:LINE-STEP(4,6),C
350 LINE(17,28)-STEP(0,4),C:LINE-STEP(-3,2),C
355 LINE-STEP(6,3),C:LINE-STEP(-7,-4),C
360 PAINT(14,29),C,C
365 LINE(40,27)-STEP(-15,15),C:LINE-STEP(-1,0),C
370 LINE-STEP(7,-7),C:LINE-STEP(3,-7),C
375 PAINT(36,29),C,C
380 LINE(37,27)-STEP(3,10),C:LINE-STEP(-8,0),C
385 LINE-STEP(0,-1),C:LINE-STEP(7,0),C
390 LINE-STEP(-3,-5),C:LINE(36,27)-STEP(3,10),C
395 GET(3,0)-(58,42),HORSE1
396 LINE(10,28)-(40,42),O,BF
400 'draw feet2
410 LINE(10,28)-STEP(-6,6),C:
  LINE-STEP(-1,6),C:LINE-STEP(1,0),C
420 LINE-STEP(1,-6),C:LINE-STEP(12,-6),C
425 PAINT(11,30),C,C:LINE(12,28)-STEP(3,8),C
430 LINE-STEP(6,3),C:LINE-STEP(1,0),C
435 LINE-STEP(-7,-3),C:LINE-STEP(2,-7),C
440 PAINT(15,31),C,C
450 LINE(40,28)-STEP(10,10),C:LINE-STEP(1,0),C
455 LINE-STEP(-10,-10),C:LINE(36,28)-STEP(14,2),C
460 LINE-STEP(-4,7),C:LINE-STEP(-1,0),C
465 LINE-STEP(4,-8),C:LINE-STEP(-7,-1),C
470 GET(3,0)-(58,40),HORSE2
471 CLS
475 PUT(0,0),HORSE1:PUT(2,0),HORSE2:
  GET(0,0)-(60,42),B1:CLS
477 PUT(0,0),HORSE2:PUT(2,0),HORSE1:
  GET(0,0)-(60,42),B2:CLS:PUT(0,30),HORSE1:PX=1
478 PUT(0,100),HORSE1
479 FOR I=0 TO 259 STEP 4:
  PUT(I,100),B1:
  ON PX GOTO 480,481
480 PUT(I,30),HORSE1:PX=2:PUT(I+4,30),HORSE2:
  GOTO 485
481 PUT(I,30),HORSE2:PX=1:PUT(I+4,30),HORSE1
485 PUT(I+2,100),B2:

```

```

        FOR J=1 TO 100:
        NEXT
490 NEXT:NEXT:END
2000 ' draw mat, jockey
2020 LINE(19,17)-(31,25),C1,BF
2030 LINE(19,17)-(31,25),0,B
2040 LINE(50,7)-(51,8),0,BF
2050 LINE(10,20)-STEP(-7,2),2:
      LINE-STEP(0,1):LINE-STEP(7,-2),2
2060 LINE(19,16)-(28,3),C
2070 CIRCLE(31,3),4,C,,,1.5
2080 LINE(30,5)-(24,14),C:LINE -STEP(8,0),C
2090 LINE-STEP(-7,8),C:LINE -STEP(2,0),C
2100 LINE-STEP(0,2),C:LINE-STEP(-5,0),C
2110 LINE-STEP(5,-8),C:LINE-STEP(-8,1),C
2120 LINE(28,10)-(35,9),C:LINE-STEP(0,1),C
2130 LINE-STEP(-9,2),C:PSET(29,5),0
2140 LINE (25,11)-(47,9),C
2150 PAINT(29,5),C,C
2160 PAINT(24,23),C,C
2170 RETURN

```

PROGRAMME 10.14

10.6 L'ANIMATION DU FOND

L'animation d'un objet peut être réalisée autrement qu'en changeant sa position sur l'écran. Si on déplace les objets se trouvant autour de lui, on aura l'impression que l'objet se déplace et non le fond. Il est très courant dans les films animés par exemple, d'avoir une voiture au centre de l'écran et un fond qui se déplace dans la direction opposée à celle du véhicule. L'illusion qui en résulte est que c'est la voiture et non le fond qui se déplace.

Dans le programme 10.5 une locomotive à vapeur se déplace le long de rails, mais ce sont les rails qui bougent et non pas la locomotive. Des lignes 30 à 90 les rails sont créés et copiés dans les tableaux M1, M2, M3 et M4. La locomotive est dessinée par les lignes 100 à 180. L'illusion du mouvement est donnée en écrivant (PUT) les rails en succession rapide sur l'écran. Ceci est réalisé par les boucles FOR des lignes 1000 à 1070. Le son produit à la ligne 1060 ajoute un peu de réalisme à l'illusion. La figure 10.9 montre la locomotive et les rails.

```

0 '** 10-15 : Train **
5 DEFINT I
10 SCREEN 1:CLS
20 N=300:DIM M1(N),M2(N),M3(N),M4(N)
30 FOR I=1 TO 4
40   FOR J=0 TO 319 STEP 16
50     LINE((I-1)*4+J,100)-
      ((I-1)*4+2+J,103),3,BF
60   NEXT
70   ON I GOTO 72,74,76,78

```



```

72     GET(0,100)-(319,102),M1:GOTO 80
74     GET(0,100)-(319,102),M2:GOTO 80
76     GET(0,100)-(319,102),M3:GOTO 80
78     GET(0,100)-(319,102),M4
80     CLS
90     NEXT
100    LINE(30,50)-(50,90),3,BF
110    LINE(50,72)-(85,90),3,BF
120    LINE(64,60)-(69,72),3,BF:'Chimney
130    LINE(33,55)-(38,70),0,BF:'Window
140    LINE(42,55)-(47,70),0,BF:'Window
150    LINE(26,50)-(30,52),3,BF
160    CIRCLE(44,88),10,2,,,1:'Big wheel
165    CIRCLE(44,88),9,2,,,1
170    CIRCLE(67,94),4,2,,,1:'Small wheel
175    CIRCLE(80,94),4,2,,,1
180    LINE(85,90)-(90,95),3
1000   FOR I=1 TO 4
1010   ON I GOTO 1020,1030,1040,1050
1020   PUT(0,100),M4,PSET:GOTO 1060
1030   PUT(0,100),M3,PSET:GOTO 1060
1040   PUT(0,100),M2,PSET:GOTO 1060
1050   PUT(0,100),M1,PSET
1060   SOUND 37+100*RND,.1
1070   NEXT:GOTO 1000

```

PROGRAMME 10.15

Pour rendre l'illusion du mouvement encore meilleure, le programme 10.16 range (GET) la locomotive dans le tableau TRAIN. A la ligne 330, avant que le son ne soit produit, un nombre au hasard entre 0 et 100 est produit. Si le nombre est inférieur à 30, la locomotive sera affichée (PUT) sur l'écran à l'emplacement (X,Y) où X est égal à 24 plus un nombre pris au hasard entre 0 et 1 et Y est égal à 48 plus un nombre au hasard entre 0 et 1. L'effet sera qu'on aura l'impression que le train est constamment en train de sauter sur les rails.

```

0  '** 10-16 : Train on bumpy track **
10  DEFINT I
20  SCREEN 1:CLS:DIM TRAIN(300)
30  N=300:DIM M1(N),M2(N),M3(N),M4(N)
40  FOR I=1 TO 4
50    FOR J=0 TO 319 STEP 16
60      LINE((I-1)*4+J,100)-
          ((I-1)*4+2+J,103),3,BF
70    NEXT
80    ON I GOTO 90,100,110,120
90      GET(0,100)-(319,102),M1:GOTO 130
100     GET(0,100)-(319,102),M2:GOTO 130
110     GET(0,100)-(319,102),M3:GOTO 130
120     GET(0,100)-(319,102),M4
130     CLS
140     NEXT
150     LINE(30,50)-(50,90),3,BF
160     LINE(50,72)-(85,90),3,BF
170     LINE(64,60)-(69,72),3,BF:'Chimney
180     LINE(33,55)-(38,70),0,BF:'Window

```

```

190 LINE(42,55)-(47,70),0,BF:'Window
200 LINE(26,50)-(30,52),3,BF
210 CIRCLE(44,88),10,2,,,1:'Big wheel
220 CIRCLE(44,88),9,2,,,1
230 CIRCLE(67,94),4,2,,,1:'Small wheel
240 CIRCLE(80,94),4,2,,,1
250 LINE(85,90)-(90,95),3
260 GET(25,49)-(91,99),TRAIN
270 FOR I=1 TO 4
280   ON I GOTO 290,300,310,320
290   PUT(0,100),M4,PSET:GOTO 330
300   PUT(0,100),M3,PSET:GOTO 330
310   PUT(0,100),M2,PSET:GOTO 330
320   PUT(0,100),M1,PSET
330   R=INT(100*RND):SOUND 37+R,.1
340   IF R<30
       THEN
           PUT(24+RND,48+RND),TRAIN,PSET
350 NEXT:GOTO 270

```

PROGRAMME 10.16

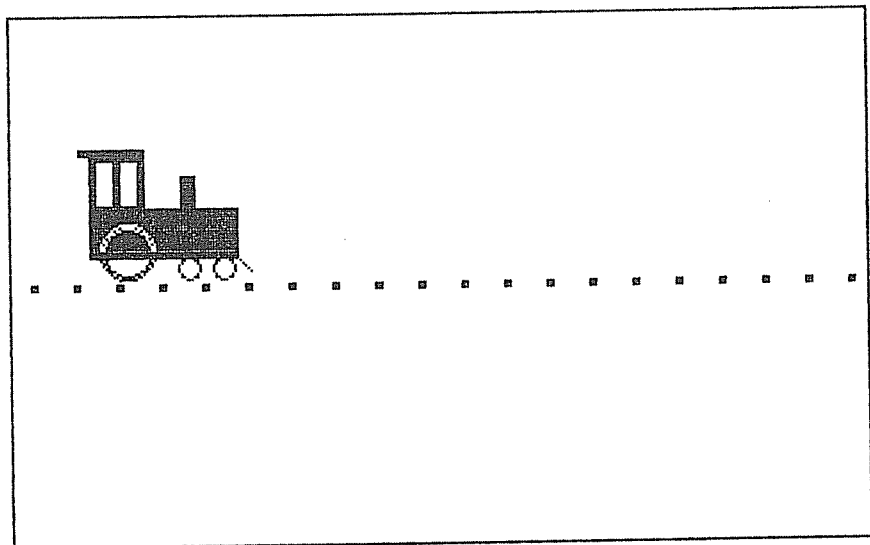


Fig. 10.9

Le dernier exemple de ce chapitre sera une fusée en route vers Mars. La fusée elle-même ne bougera pas et les étoiles bougeront pour donner l'illusion du mouvement. Pour animer un peu la scène, la traînée d'étincelles laissée par la fusée sera visible et en mouvement constant. Les lignes 30 à 180 du programme 10.17 créent quatre dessins aléatoires de points qui seront mémorisés dans les tableaux M1, M2, M3, M4 et M5. Ces points seront affichés (PUT) en arrière de la fusée, en mode PSET pour simuler une traînée d'étincelles. La fusée est dessinée par les lignes 190 à 300.

Aux lignes 310 à 340, les coordonnées de huit étoiles sont calculées aléatoirement. Si le nombre généré pour la coordonnée horizontale d'une étoile tombe entre 143 et 177, il est rejeté, ou sinon l'étoile traversera la fusée et en effacera une partie. La boucle des lignes 400 à 530 affiche (PUT) les traînées d'étincelles, et met à jour les huit étoiles à chaque itération. Le résultat donne une bonne impression de mouvement. La figure 10.10 montre la fusée avec les étoiles et un des modèles d'étincelles.

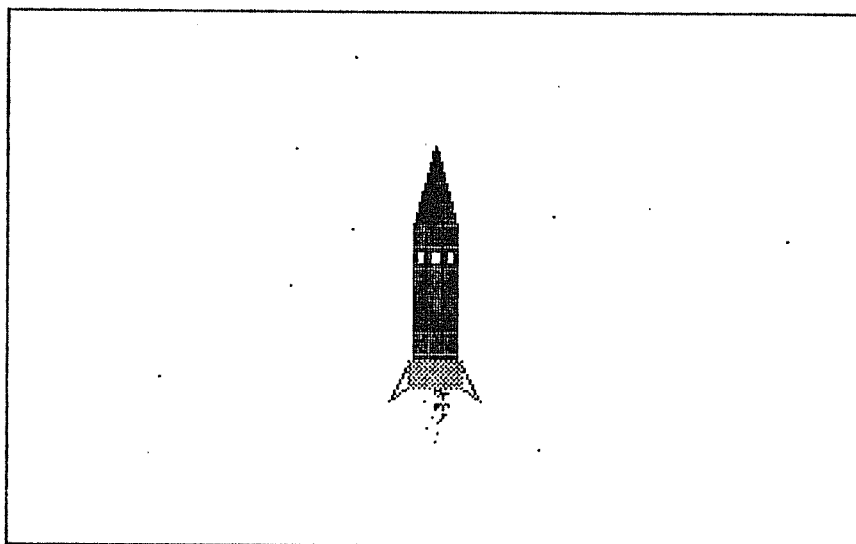


Fig. 10.10

```

0  '** 10-17 : Rocket to Mars **
5  DEFINT A-Z: DIM X(20), Y(20)
10  SCREEN 1: COLOR 0,0: CLS: N=200
20  DIM M1(N), M2(N), M3(N), M4(N), M5(N)
30  FOR I=1 TO 5
40    FOR J=1 TO 50
50      R1=8*RND: R2=20*RND
60      ON 2*RND GOTO 70,90
70      PSET (20+R1*RND, R2*RND), 1+2*RND
80      GOTO 100
90      PSET (20-R1*RND, R2*RND), 1+2*RND
100    NEXT
110    ON I GOTO 120,130,140,150,160
120    GET (10,0)-(30,20), M1: GOTO 170
130    GET (10,0)-(30,20), M2: GOTO 170
140    GET (10,0)-(30,20), M3: GOTO 170
150    GET (10,0)-(30,20), M4: GOTO 170
160    GET (10,0)-(30,20), M5
170    CLS
180  NEXT

```

```

190 LINE(152,80)-(168,130),1,BF
200 LINE(152,80)-(160,50),1
210 LINE-(168,80),1
220 PAINT(160,70),1,1
230 LINE(150,130)-(170,140),2,BF
240 LINE(150,132)-(143,145)
250 LINE-(150,140)
260 LINE(170,132)-(177,145)
270 LINE-(170,140)
280 LINE(154,90)-(155,93),0,BF
290 LINE(159,90)-(161,93),0,BF
300 LINE(165,90)-(166,93),0,BF
310 FOR I=1 TO 8
320   X=319*RND:
      IF X>143 AND X<177
      THEN
        320
330   X(I)=X:Y(I)=199*RND
340 NEXT
350 G=1
400 ON G GOTO 410,420,430,440,450
410 PUT(150,141),M1,PSET:GOTO 460
420 PUT(150,141),M2,PSET:GOTO 460
430 PUT(150,141),M3,PSET:GOTO 460
440 PUT(150,141),M4,PSET:GOTO 460
450 PUT(150,141),M5,PSET
460 FOR I=1 TO 8
470   Y=Y(I)+4:
      IF Y>199
      THEN
        Y=3
480   PRESET(X(I),Y(I))
490   PSET(X(I),Y)
500   Y(I)=Y
510 NEXT
520 G=G+1:
      IF G>5
      THEN
        G=1
530 GOTO 400

```

PROGRAMME 10.17

10.7 L'ANIMATION PHOTOGRAPHIQUE

Pour produire une bonne animation, les programmes doivent effectuer un grand nombre d'opérations et de comparaisons. Quand les graphiques deviennent compliqués et les objets croissent en taille, l'animation devient moins efficace. Il est impensable, par exemple, d'écrire en BASIC un programme dans lequel un fond qui couvre tout l'écran se déplace rapidement pour répondre à quelque action. BASIC n'est pas assez rapide.

Depuis le début du siècle, l'animation a été réalisée par des techniques cinématographiques. Avec celles-ci des images peuvent être photographiées une par une, et juxtaposées à grande vitesse au moment de la projection

(les projecteurs courants montrent des films de 16 ou 24 images par seconde) pour produire l'illusion de mouvement.

Pour photographier des images qui ne sont pas créées instantanément, il est nécessaire de disposer de cameras « pas à pas » (ayant la possibilité de prendre une image à la fois). Pour éviter des images doubles produites par des reflets, la zone derrière l'écran vidéo doit être aussi sombre que possible. A cause du principe de balayage de l'écran vidéo, il est nécessaire de filmer lentement (en général moins de 1/30ème ou 1/15ème de seconde). Dans la plupart des caméras la vitesse de l'obturateur est contrôlée automatiquement par la quantité de lumière disponible, ainsi le diaphragme le plus petit possible sera utilisé. Egalement, comme la position de la caméra par rapport au moniteur doit rester constante, un trépied fixe est nécessaire.

Tout un monde nouveau s'ouvre avec l'animation photographique : des objets tri-dimensionnels peuvent tourner et se déplacer sur l'écran et des fonds aussi complexes soient-ils peuvent être utilisés, parce qu'alors il n'est pas nécessaire d'obtenir les images en une succession rapide. Dans les films animés par ordinateur, certaines images ont pris des heures pour être générées !

EXERCICES

1. Ecrire un programme qui simule une balle rebondissante sur les quatre murs de l'écran.
2. Ecrire un programme dans lequel la balle rebondissante du programme 10.1 démarre avec une vitesse égale à 0, accélère lentement (tout en rebondissant entre les murs) jusqu'à ce qu'elle atteigne une vitesse maximum et décélère jusqu'à ce qu'elle s'arrête.
3. Prendre les trois figures du programme 10.9 et créer l'image miroir, de telle sorte que l'homme puisse marcher en arrière et en avant. Utiliser la méthode montrée au paragraphe 8.5.
4. Ecrire un programme dans lequel une voiture de course soit vue de dessus et où l'illusion de mouvement soit créée par les côtés de la route se déplaçant vers le bas. La voiture devra se déplacer à gauche et à droite à l'intérieur de la piste, guidée par les touches de contrôle du curseur du bloc de touches numériques. Quand elle touche le côté de la route elle doit s'arrêter.

Graphiques tridimensionnels

Les dix chapitres précédents étaient consacrés à la manipulation d'images à deux dimensions, ce qui est la chose la plus logique si on considère que l'écran est plat de même que le papier sur lequel on transfère généralement les graphiques. La majorité des tableaux de ce qu'on a appelé la période classique étaient des tentatives pour figer une scène, un moment, de la manière la plus réaliste. Comme notre réalité est tridimensionnelle, la profondeur joua un rôle majeur dans cet effort pour saisir la réalité sur une toile ou un autre matériau. Chaque artiste se rend compte que ce n'est pas une tâche facile. Heureusement pour ceux d'entre nous qui ne sont pas nés avec des dons spéciaux pour le dessin, un grand nombre de techniques et de détails nécessaires pour transférer un objet solide sur une surface plane ont été analysés par des mathématiciens et des ingénieurs, et tout un ensemble de formules et de fonctions nous permet de confier à l'ordinateur les calculs de volume, de perspective, et de point de vue.

Le système de coordonnées cartésiennes X - Y que nous avons utilisé si fréquemment est idéal pour décrire des formes planes, mais il est inadéquat pour des objets ou des fonctions solides. Pour travailler avec des graphiques tridimensionnels, nous devons avoir recours à un système à trois axes, que nous appellerons X , Y et Z . La figure 11.1 montre une des orientations possibles de ce système.

Dans les coordonnées cartésiennes planes, chaque point était identifié par un couple de nombres, le premier indiquant la colonne et le second la ligne ¹. Avec ce nouveau système de trois axes, chaque point doit être identifié par un triplet, c'est-à-dire trois nombres dans lesquels les deux

premiers sont ceux que nous avons utilisés précédemment, colonne et ligne, et le troisième indique la distance du point au plan X–Y. Une manière d'imaginer ce système est de penser que les points ne se trouvent pas seulement sur l'écran, mais aussi à l'extérieur et à l'intérieur de celui-ci ; c'est-à-dire que la coordonnée Z est la profondeur par rapport au plan X–Y, dans notre cas l'écran. La figure 11.2 montre la position du point (4,3,1) dans le système à trois axes. On peut l'imaginer comme un point situé à la quatrième colonne et à la troisième ligne et une unité devant l'écran, flottant dans l'air.

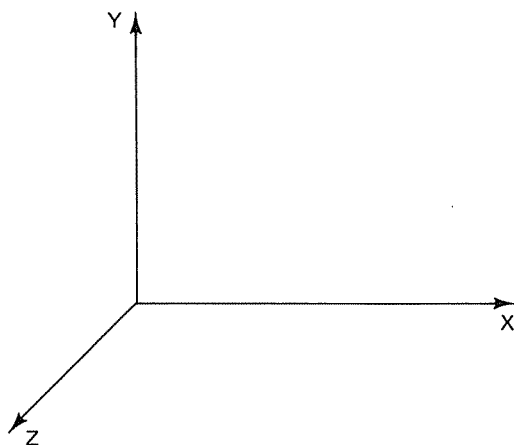


Fig. 11.1

Ce chapitre est divisé en deux parties à peu près indépendantes : le tracé de fonctions tridimensionnelles et la représentation d'objets solides.

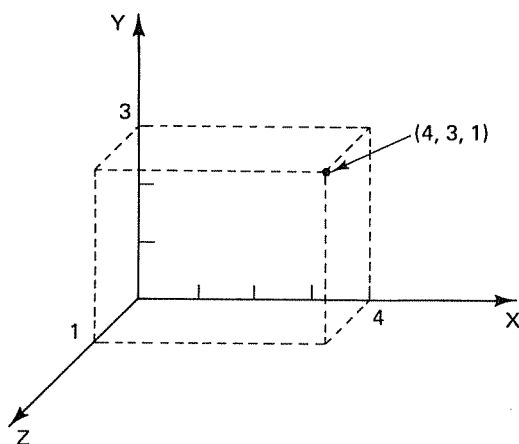


Fig. 11.2

11.1 FONCTIONS TRIDIMENSIONNELLES

Les fonctions qui ont été tracées aux paragraphes 1.6 et 2.5 sont appelées habituellement fonctions en R^2 , fonctions à deux variables ou fonctions bidimensionnelles. Leur caractéristique est que, à chaque valeur de la variable indépendante (que nous avons d'habitude appelée X) correspond une valeur unique de la variable dépendante, communément appelée Y.

Les fonctions tridimensionnelles (que nous appellerons aussi fonctions 3-D) ont deux variables indépendantes (que nous appellerons X et Y) et une dépendante (que nous appellerons Z). La fonction 11.1 est un exemple de ce type de fonction.

$$F(X,Y)=\text{SIN}(X)^*\text{SIN}(Y). \quad (11.1)$$

A chaque point (X,Y), la fonction affecte une valeur unique, le produit des sinus des deux valeurs. Bien que ces fonctions génèrent des courbes qui sont seulement représentables dans l'espace, elles peuvent être projetées et vues sur des surfaces à deux dimensions, de la même manière que nous représentons un objet solide sur un dessin plat ou une photographie. La figure 11.3 montre la fonction 11.1 de $X = -1,5$ à $X = 7,7$ et de $Y = 1,9$ à $Y = 11,9$. Comme vous pouvez voir, bien que le tracé soit fait sur une surface plane (le papier dans ce cas), le graphique donne une impression très nette de volume.

Les axes X et Y déterminent un plan (appelé le plan X-Y), et à chaque point de ce plan (c'est-à-dire à chaque couple X-Y) la fonction affecte une valeur unique, la hauteur suivant l'axe Z, ou la distance au-dessus (ou au-dessous) du plan X-Y. La fonction 11.2 affecte la valeur 1 à chaque point X, Y.

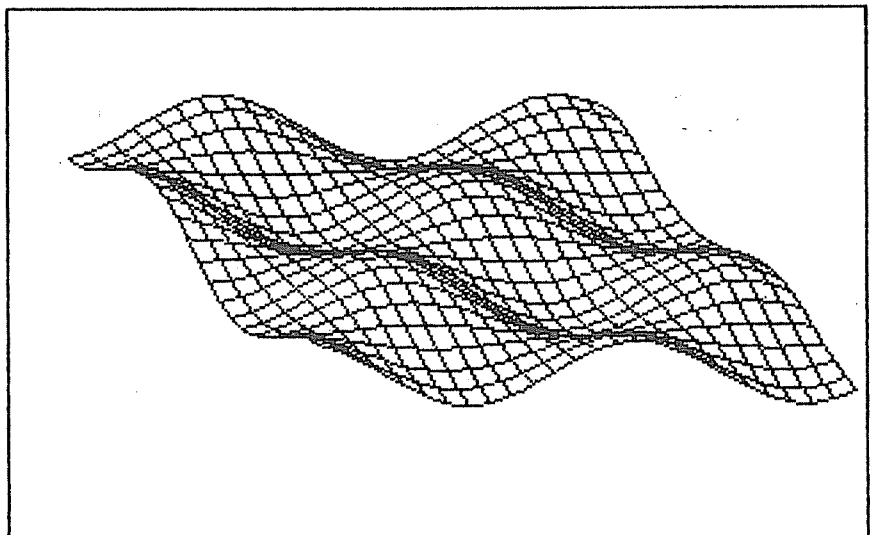


Fig. 11.3

Supposez que nous choisissons le point (4,3) du plan X-Y. Pour trouver sa hauteur (ou coordonnée Z) nous donnons ces deux valeurs à la fonctions 11.2, que l'on a appelée FN HEIGHT à la ligne 10 du programme 11.1.

```
0  °° 11-1 : A 3-D function °°
10 DEF FN HEIGHT(X,Y)=1
20 X=4:Y=3:Z=FN HEIGHT(X,Y)
30 PRINT X,Y,Z
```

PROGRAMME 11.1

La figure 11.2 montre la position de ce point dans le système à trois coordonnées. Comme la fonction 11.2 donne 1 pour chaque paire (X,Y), la représentation tridimensionnelle de la fonction est un plan parallèle au plan X-Y et placé exactement une unité au-dessus de lui.

11.1.1 La perspective droite

Nous allons commencer à tracer une fonction 3-D sur l'écran graphique en utilisant la méthode la plus simple. Vous vous rappelez que la coordonnée Z de la fonction 11.1 était donnée par la formule $Z = \sin(X) * \sin(Y)$. Le sinus de 1,571 est approximativement 1, aussi nous allons fixer X égal à 1,571 et le garder constant pour simplifier cet exemple. Si nous faisons varier Y entre certaines valeurs, la fonction renverra la valeur $Z = 1 * \sin(Y)$, qui est simplement $\sin(Y)$. La figure 11.4 montre la figure résultante avec le plan X-Y au niveau de l'œil, c'est-à-dire que l'axe des X pointe directement en dehors de la feuille, dans la direction de l'œil.

Si nous comparons cette représentation avec celle d'une fonction bidimensionnelle normale, la seule différence apparente est que l'axe des Z a remplacé l'axe des Y, et l'axe des Y a remplacé l'axe des X. Ce qui se passe, c'est que la figure 11.4 est une tranche de l'image complète, une coupe sélective du plan Y-Z à $X = 1,571$. Chaque point tracé n'est pas seulement composé d'un couple de nombres (Y,Z), mais est en fait un triplet (1,571,Y,Z).

Si nous fixons X égal à 0,5236 (le sinus de 0,5236 est approximativement 0,5) et utilisons pour Y les mêmes valeurs que nous avons utilisées pour tracer la figure 11.4, chaque valeur Z sera définie par $Z = 0,5 * \sin(Y)$, et ainsi la courbe aura une hauteur moitié de celle de la figure 11.4. Dans la figure 11.5 la ligne continue est tracée avec $X = 0,5236$ et celle en pointillé avec $X = 1,571$.

Pour tracer la fonction, nous considérerons plusieurs coupes successives, en prenant différentes valeurs de X, et dessinerons les courbes résultantes l'une au-dessus de l'autre. Dans le programme 11.2, chaque valeur de X est

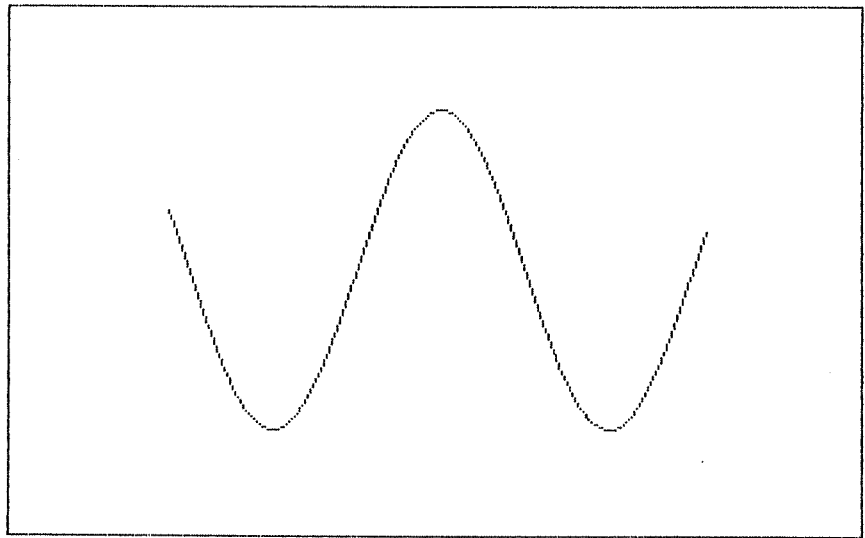


Fig. 11.4

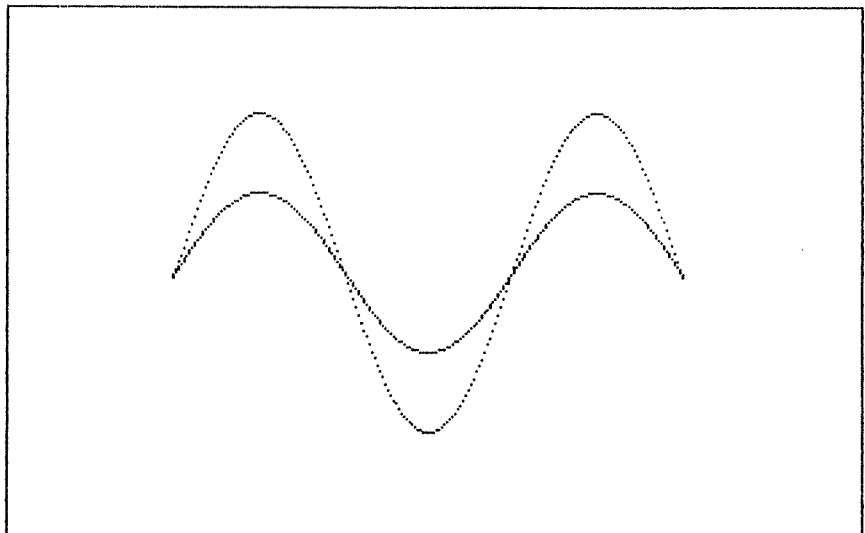


Fig. 11.5

multipliée par 0,1 et chaque valeur de Y par 0,05, pour obtenir la bonne fréquence (voir paragraphe 1.6). Chaque valeur de Z est multipliée par 60 pour avoir la bonne amplitude. La fonction est tracée entre $X = -1,5$ et $X = -0,1$, et $Y = 2,9$ et $Y = 12,95$ comme le montre la figure 11.6.

```

0 '** 11-2 : Straight perspective **
10 DEF FN HEIGHT(X,Y)=60*SIN(X*.1)*SIN(Y*.05)
20 SCREEN 1:CLS
30 FOR X=-15 TO 0 STEP 2
40   FOR Y=58 TO 259
50     Z=FN HEIGHT(X,Y)
60     PSET(Y,100-Z)
70   NEXT
80 NEXT

```

PROGRAMME 11.2

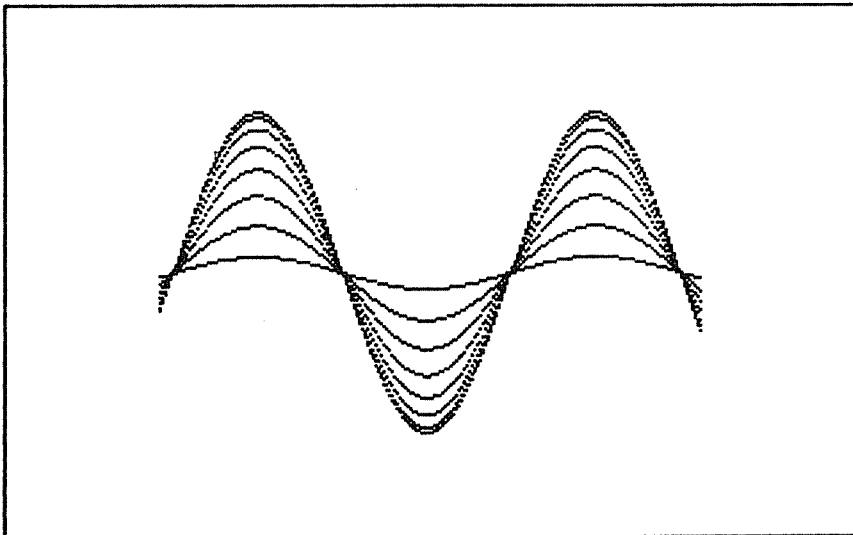


Fig. 11.6

Du fait que toutes les courbes se superposent, la figure résultante ne présente pas vraiment l'effet de volume recherché. Qui plus est, si on prend une fourchette de valeurs X plus large, un encore plus grand nombre de lignes seront tracées, rendant alors le dessin très confus. Changez la valeur de fin de la boucle FOR (ligne 30) en 15 et regardez le graphique résultant.

11.1.2 Inclinaison du plan X-Y

Une méthode pour rendre la représentation d'une fonction tridimensionnelle plus compréhensible est d'ajouter une valeur à la coordonnée de hauteur (dans ce cas la coordonnée Z), ainsi chaque courbe est tracée en dessous de la précédente, ce qui entraîne une inclinaison du plan X-Y par rapport à l'écran. De cette manière la plus grande partie du tracé de chaque courbe n'entrera pas en conflit avec les autres. Le pro-

gramme 11.3 ajoute X à la coordonnée Z. L'amplitude de la fonction a été réduite pour minimiser les superpositions. Le résultat est l'image de la figure 11.7. Comparée avec celle de la figure 11.3 qui montre la même fonction, il s'agit ici d'une vue de face.

```
0 '** 11-3 : Inclined plane **
10 DEF FN HEIGHT(X,Y)=10*SIN(X*.1)*SIN(Y*.05)
20 SCREEN 1:CLS
30 FOR X=-15 TO 80 STEP 4
40   FOR Y=58 TO 259
50     Z=FN HEIGHT(X,Y)
60     IF Y=58
70       THEN
80         PSET(Y,60-Z+X)
90       ELSE
100        LINE-(Y,60-Z+X)
110   NEXT Y
120 NEXT X
```

PROGRAMME 11.3

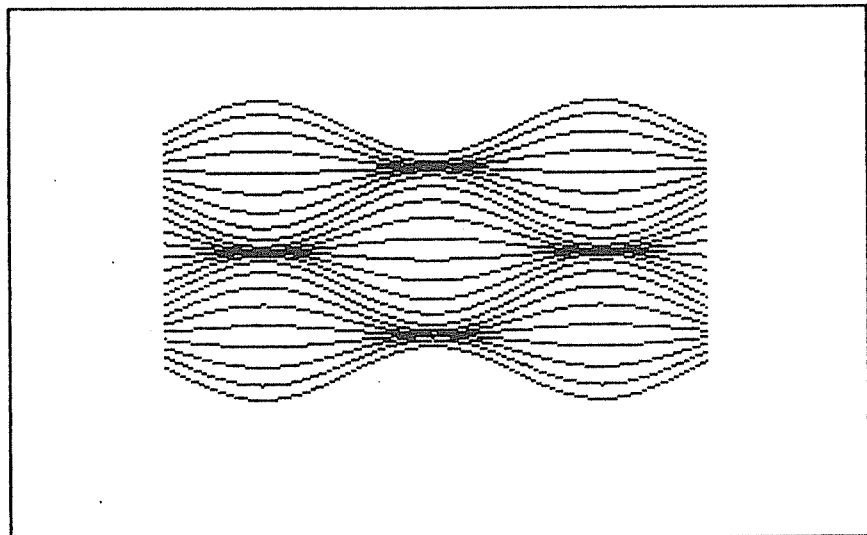


Fig. 11.7

11.1.3 Vue de côté

Un moyen d'améliorer encore la représentation de fonctions 3-D est d'en montrer une vue de côté, au lieu d'une vue droite ou de face. La méthode la plus simple consiste à tracer les courbes correspondant à chaque valeur de X, à la fois en dessous et à droite (ou à gauche) de la précédente. Au programme 11.4 nous ajoutons X à la coordonnée horizontale, dans ce cas Y.

```

0  '** 11-4 : Lateral view **
10 DEF FN HEIGHT(X,Y)=12*SIN(X*.1)*SIN(Y*.05)
20 SCREEN 1:CLS
30 FOR X=-15 TO 80 STEP 4
40   FOR Y=38 TO 239 STEP 4
50     Z=FN HEIGHT(X,Y)
60     IF Y=38
        THEN
            PSET(Y+X,60-Z+X)
        ELSE
            LINE-(Y+X,60-Z+X)
70   NEXT Y
80 NEXT X

```

PROGRAMME 11.4

La figure 11.8 montre le graphique tracé par le programme 11.4

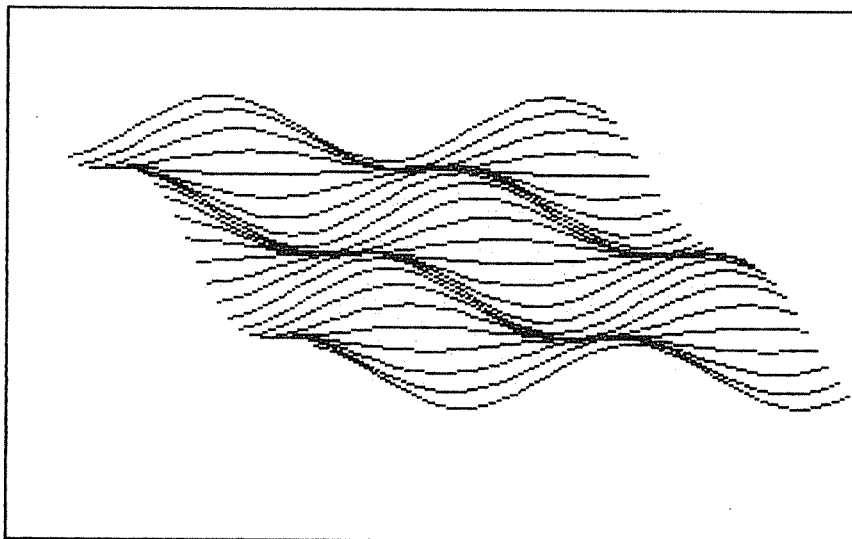


Fig. 11.8

11.1.4 Tracé de grilles

Considérez une fonction 3-D comme un objet solide mais transparent. Une manière de le rendre visible serait d'en matérialiser les contours à l'aide de fils parallèles : c'est ce que nous avons fait jusqu'ici. Une autre méthode (qui conduit quelquefois à un dessin plus clair) serait de placer un filet très flexible et très lourd (fait peut-être de fils de caoutchouc) au-dessus de la fonction de telle sorte que le profil du terrain serait matérialisé suivant deux directions.

Dans les programmes précédents, nous fixions une valeur de X et balayions l'ensemble des valeurs Y. Nous choisissons alors une autre valeur de X et recommençons l'opération. Pour obtenir un tracé perpendiculaire et compléter ainsi la grille, tout ce que nous avons à faire est de fixer des valeurs de Y et balayer l'ensemble des valeurs de X.

Dans le programme 11.5, le pas de la boucle FOR de variation de Y (ligne 130) a été augmenté, sinon le maillage serait trop fin et le résultat confus. On a déjà vu le graphique résultant figure 11.3.

```

0  '** 11-5 : Function with grid **
10 DEF FN HEIGHT(X,Y)=12*SIN(X*.1)*SIN(Y*.05)
20 SCREEN 1:CLS
30 FOR X=-15 TO 80 STEP 4
40   FOR Y=38 TO 239 STEP 4
50     Z=FN HEIGHT(X,Y)
60     IF Y=38
70       THEN
80         PSET(Y+X,60-Z+X)
90       ELSE
100        LINE-(Y+X,60-Z+X)
110      NEXT Y
120    NEXT X
130  FOR Y=38 TO 239 STEP 8
140    FOR X=-15 TO 80 STEP 4
150      Z=FN HEIGHT(X,Y)
160      IF X=-15
170        THEN
180          PSET(Y+X,60-Z+X)
190        ELSE
200         LINE-(Y+X,60-Z+X)
210      NEXT X
220    NEXT Y
230 NEXT Y

```

PROGRAMME 11.5

11.1.5 La suppression des lignes cachées

Les méthodes utilisées jusqu'ici pour tracer des fonctions 3-D produisent un effet semblable à des modèles en fil de fer, que ce soit avec un maillage ou avec des lignes parallèles. Bien que ces méthodes conviennent pour certaines fonctions simples, il devient difficile de distinguer quoi que ce soit lorsque plusieurs surfaces se chevauchent. Nous avons pris garde de ne pas fixer de trop grandes amplitudes pour éviter des conflits. La figure 11.9 montre le résultat obtenu en changeant la ligne 10 du programme 11.5 en :

```

10 DEF FN HEIGHT(X,Y)=40*SIN(X*.1)*SIN(Y*.05)

```

L'impression de troisième dimension est plus sensible avec une grande amplitude, mais, plusieurs parties se recouvrant, la fonction devient difficile à visualiser. Une solution partielle consiste à réduire l'incrément

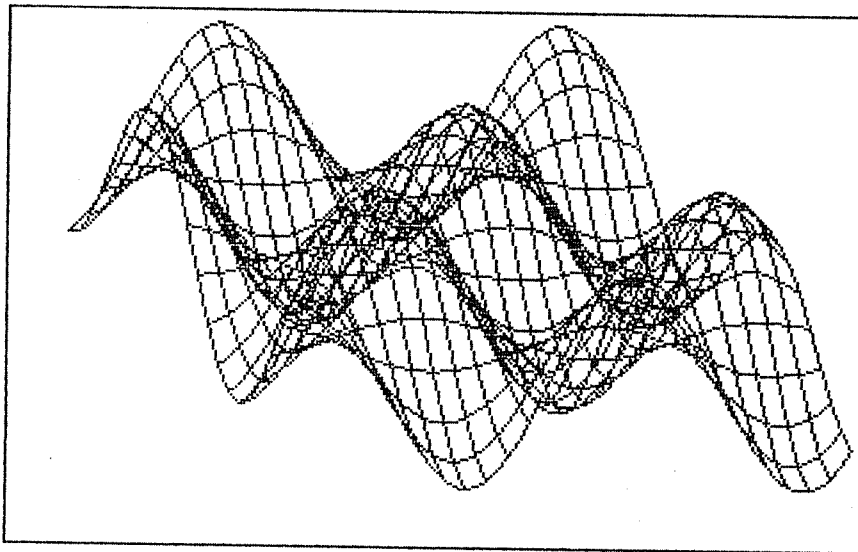


Fig. 11.9

vertical de sorte que chaque nouvelle ligne soit tracée à une distance plus grande et le nombre d'intersections minimisé. Malheureusement cela diminue aussi l'apparence de volume. Aux lignes 60 à 160 du programme 11.6 nous avons ajouté $2 \cdot X$ (contrairement à X lors des précédents programmes) à l'incrément vertical des courbes individuelles. La figure 11.10 montre le graphique résultant.

```

0  '** 11-6 : Increased inclination **
10 DEF FN HEIGHT(X,Y)=25*SIN(X*.1)*SIN(Y*.05)
20 SCREEN 1:CLS
30 FOR X=-15 TO 65 STEP 4
40   FOR Y=38 TO 239 STEP 4
50     Z=FN HEIGHT(X,Y)
60     IF Y=38
70       THEN
80         PSET(Y+X,60-Z+2*X)
90       ELSE
100        LINE-(Y+X,60-Z+2*X)
110      NEXT
120    NEXT
130  FOR Y=38 TO 239 STEP 8
140    FOR X=-15 TO 65 STEP 4
150      Z=FN HEIGHT(X,Y)
160      IF X=-15
170        THEN
180          PSET(Y+X,60-Z+2*X)
190        ELSE
200          LINE-(Y+X,60-Z+X*2)
210        NEXT
220      NEXT
230    NEXT
240  NEXT

```

PROGRAMME 11.6

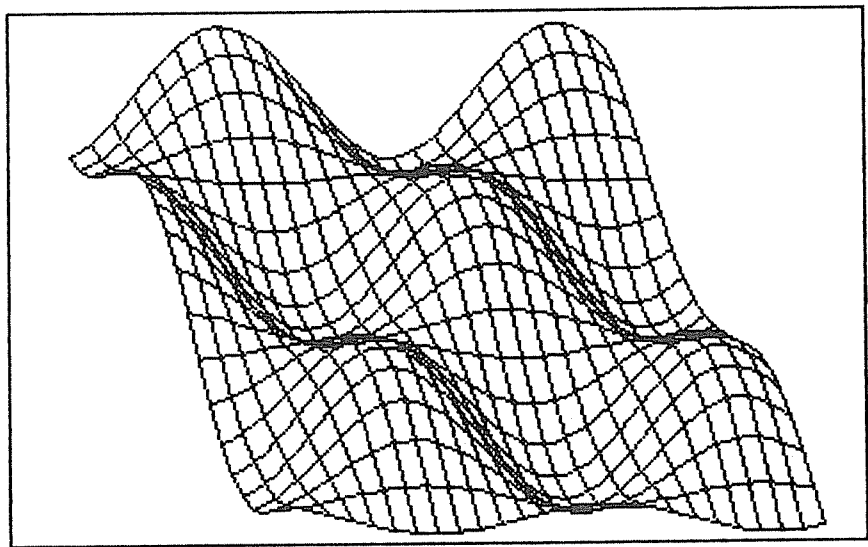


Fig. 11.10

Dans ce paragraphe nous éliminerons les lignes qui sont cachées par l'avant-plan, lignes qui seraient invisibles si la surface était solide et opaque. Pour faciliter le processus, nous allons retourner à la méthode des lignes parallèles et mettre de côté pour l'instant la méthode du maillage.

Comme la fonction est tracée comme si on la voyait de face et de dessus et que le processus commence de l'arrière vers l'avant, chaque nouvelle ligne tracée est plus proche de l'observateur et par conséquent couvre tous les points ou lignes situés en dessous d'elle. C'est ainsi que cela se passe dans la vie courante : quand vous regardez un objet du dessus, toute partie opaque cache ce qui se situe au-dessous de la ligne de vision. Retournons à nos fonctions : quand des points ou des lignes se retrouvent situés au-dessous de la dernière ligne générée sur l'écran, cela signifie qu'ils ne doivent pas être vus. Assurez-vous que vous avez bien compris ceci.

Partant de ce fait, nous pouvons tout effacer à partir du point en cours jusqu'au bord inférieur de l'écran pour nous débarrasser des lignes cachées, ce qui est accompli par le programme 11.7, avec l'instruction `LINE(NX,NY)-(NX,199)`. Comme cette instruction laisse le LRP à `(NX,199)`, nous ne pouvons plus utiliser l'instruction `LINE-(NX,NY)` dont nous nous sommes servis pour tracer une ligne continue. Nous aurons recours à une alternative exposée au paragraphe 2.5 : on mémorise les coordonnées du dernier point calculé dans `PX` et `PY` et les lignes sont tracées de `(PX,PY)` à `(NX,NY)`.

```
0 '** 11-7 : Hidden lines removed **
10 DEF FN HEIGHT(X,Y)=
    25*SIN(X*.15)*SIN(Y*8.999999E-02)
```



```

20 SCREEN 1:CLS
30 FOR X=-15 TO 65 STEP 2
40   FOR Y=38 TO 239
50     Z=FN HEIGHT(X,Y)
60     NY=Y+X:NZ=60-Z+X
70     LINE(NY,NZ)-(NY,199),0
80     IF Y=38
        THEN
          PSET(NY,NZ)
        ELSE
          LINE(PY,PZ)-(NY,NZ)
90     PY=NY:PZ=NZ
100   NEXT Y
110 NEXT X

```

PROGRAMME 11.7

La figure 11.11 montre la fonction avec les lignes cachées supprimées.

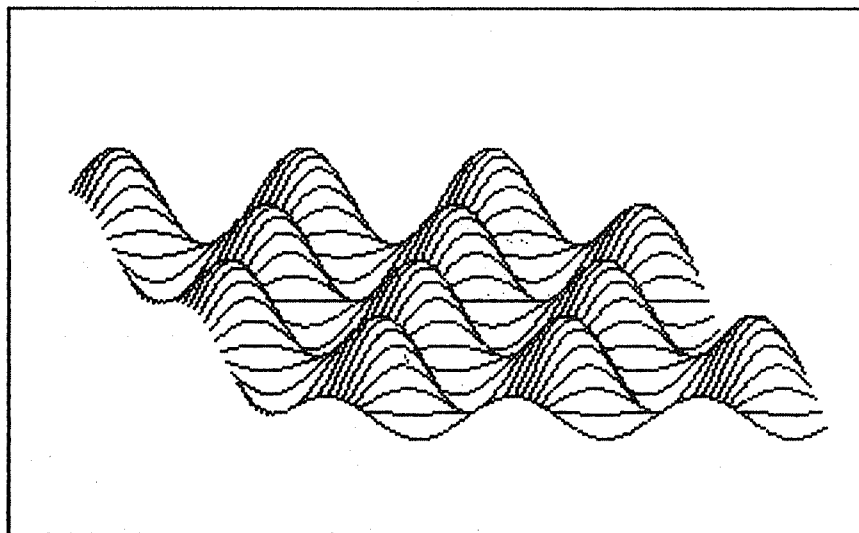


Fig. 11.11

Le pas de boucle FOR pour Y à la ligne 40 doit être 1 pour garantir l'effacement de toutes les lignes cachées. Essayez le programme 11.7 avec un incrément supérieur à 1.

11.1.6 Comment voir les parties les plus basses

Quand la fonction présente des maximums d'amplitude pour les valeurs de X élevées (c'est-à-dire pour les dernières lignes tracées vers le bas de l'écran) il peut arriver que certaines parties importantes soient couvertes et perdues. Le cas extrême est représenté par une fonction où la dernière valeur de X dessine une ligne au-dessus de toutes les précédentes, auquel cas elle reste seule présente sur l'écran.

La figure 11.12 montre le graphique réalisé par le programme 11.8. Bien qu'il soit très clair, une partie importante de la fonction a été laissée de côté.

```

0 '** 11-8 : Erases bottom surface **
10 SCREEN 1:COLOR 0,0:CLS:DIM M(4001)
20 DEF FN H(X,Y)=20*COS(SQR(X*X+Y*Y))
30 FOR X=-30 TO 30 STEP 4
40   FOR Y=-100 TO 100
50     GOSUB 1000:Z=H
60     NY=Y+160:NZ=100-Z*.5
70     LINE(NY,NZ)-(NY,199),0
80     IF Y>-100
        THEN
          LINE(PY,PZ)-(NY,NZ)
90     PY=NY:PZ=NZ
100   NEXT
110 NEXT
120 END
1000 X1=X*.2:Y1=Y*.2
1010 D=SQR(X1*X1+Y1*Y1)
1020 H=20*COS(D)*1/(D*.1)
1030 RETURN

```

PROGRAMME 11.8

La figure 11.13 montre le graphique produit quand le programme est arrêté après X=10. Il est clair que si on le laisse continuer, une partie importante sera effacée.

Pour résoudre ceci, deux graphiques séparés doivent être dessinés : l'un vu de dessus (celui des paragraphes précédents) et un second vu de dessous. Quand on combinera ces deux graphiques, la fonction sera visible dans sa totalité.

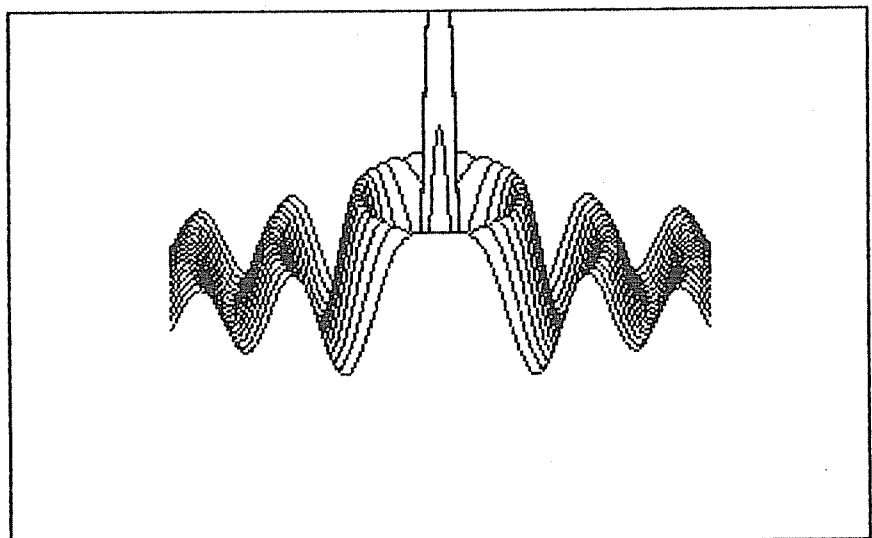


Fig. 11.12

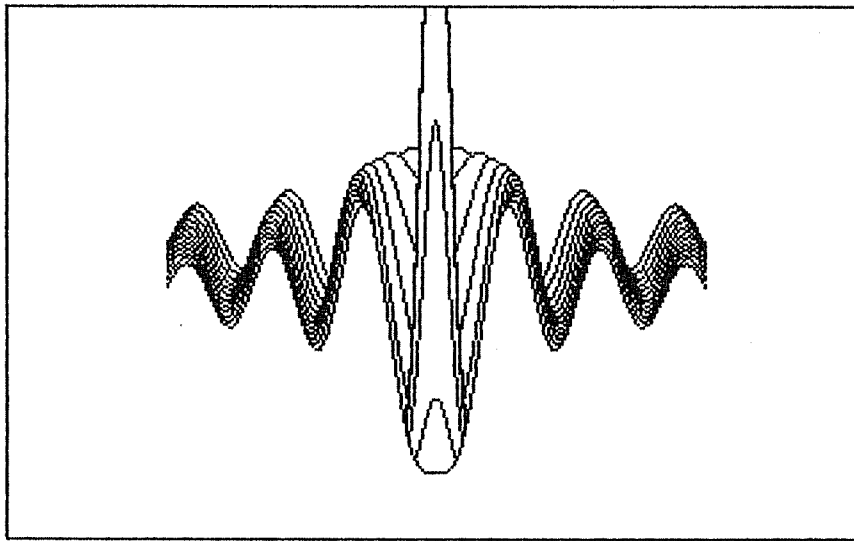


Fig. 11.13

La vue de dessous est obtenue de la même manière que la vue de dessus, mais la ligne effaçante est tracée à partir du point en cours jusqu'au haut de l'écran. Si par exemple le point à tracer est (Y,Z) , l'instruction sera $LINE(Y,Z)-(Y,0),0$. Les lignes 30 à 110 du programme 11.9 tracent la vue de dessus dans la couleur 3 (la partie que l'on voit sur la figure 11.12) et la totalité de l'écran est sauvegardée dans le tableau M avec l'instruction GET de la ligne 120. Pour tracer la seconde partie, l'écran est effacé et aux lignes 130 à 210 la vue de dessous est dessinée dans la couleur 2. Le tableau M est restitué (PUT) sur l'écran sous le mode OR et les deux images sont combinées.

```

0  '** 11-9 : Shows bottom surface **
10 SCREEN 1:COLOR 0,0:CLS:DIM M(4000)
20 DEF FN H(X,Y)=20*COS(SQR(X*X+Y*Y))
30 FOR X=-30 TO 30 STEP 4
40   FOR Y=-100 TO 100
50     GOSUB 1000:Z=H
60     NY=Y+160:NZ=100-Z+X*.5
70     LINE(NY,NZ)-(NY,199),0
80     IF Y>-100
90       THEN
100        LINE(PY,PZ)-(NY,NZ)
110        PY=NY:PZ=NZ
120 GET(0,0)-(319,199),M:CLS
130 FOR X=-30 TO 30 STEP 4
140   FOR Y=-100 TO 100
150     GOSUB 1000:Z=H
160     NY=Y+160:NZ=100-Z+X*.5
170     LINE(NY,NZ)-(NY,0),0

```

```

180      IF Y>-100
          THEN
              LINE (PY,PZ)-(NY,NZ),2
190      PY=NY:PZ=NZ
200      NEXT
210 NEXT
220 PUT (0,0),M,OR
230 W$=INPUT$(1):END
1000 X1=X*.2:Y1=Y*.2
1010 D=SQR(X1*X1+Y1*Y1)
1020 H=20*COS(D)*1/(D*.1)
1030 RETURN

```

PROGRAMME 11.9

On voit la vue du fond en figure 11.14, et le dessin final, combinaison des figures 11.12 et 11.14 en figure 11.15.

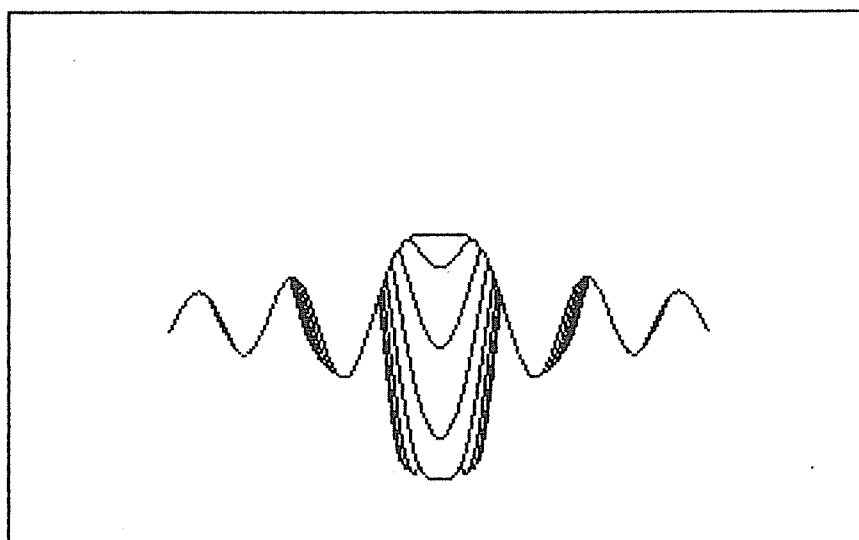


Fig. 11.14

11.1.7 Grille avec les lignes cachées supprimées

Dans certains cas, la qualité du graphique s'améliore considérablement si les courbes X et Y sont toutes les deux tracées (pour produire l'effet « filet »). Malheureusement, quand on utilise la technique de suppression des lignes cachées, la deuxième partie (quand on traverse la fonction avec des valeurs de Y croissantes) de l'image efface toujours la première (la partie qui traverse la fonction avec les valeurs de X).

Il est possible de superposer les deux grilles avec la procédure suivante :

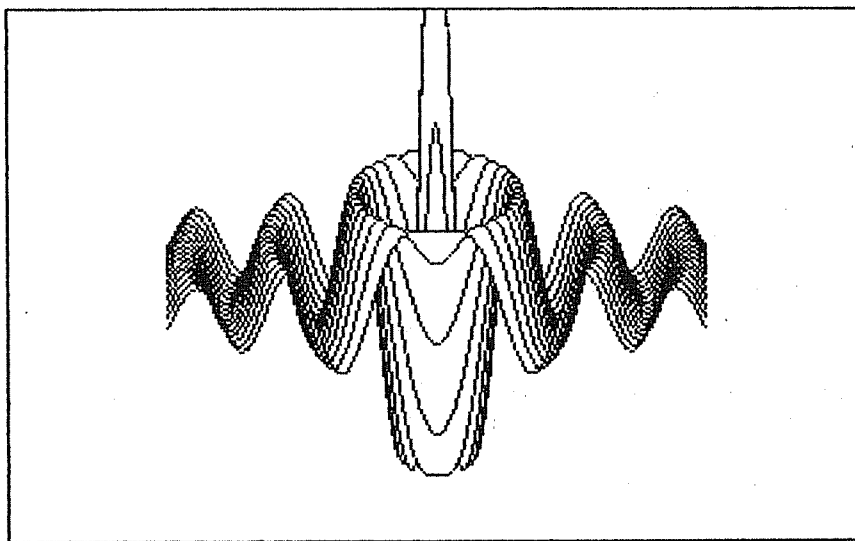


Fig. 11.15

Dessiner la première partie du graphique en fixant les valeurs X et en effaçant les lignes cachées.

Ranger (GET) toute l'image dans un tableau.

Effacer l'écran.

Dessiner la seconde partie en fixant les valeurs Y et en effaçant aussi les lignes cachées.

Ecrire (PUT) la première image sous le mode OR.

Les lignes 30 à 110 du programme 11.10 tracent la fonction définie aux lignes 1000 à 1030 pour différentes valeurs de X. La figure 11.16 montre le graphique résultant.

```

0  '** 11-10 : Grid **
10 DEFINT X,Y
20 SCREEN 1:CLS:DIM M(4000)
30 FOR X=-50 TO 50 STEP 3
40   FOR Y=-100 TO 100
50     GOSUB 1000
60     NY=Y+X*.5+160:NZ=90+Z+X
70     LINE(NY,NZ)-(NY,199),0
80     IF Y>-100
90       THEN
100        LINE(PY,PZ)-(NY,NZ)
110        PY=NY:PZ=NZ
120      NEXT
130    NEXT
140  NEXT
150  LINE(0,0)-(319,199),3,B
160  GET(0,0)-(319,199),M:CLS
170  FOR Y=100 TO -100 STEP-10
180    FOR X=-50 TO 50
190      GOSUB 1000

```

```

260     NY=Y+X*.5+160:NZ=90+Z+X
270     LINE(NY,NZ)-(NY,199),0
280     IF X>-50
        THEN
            LINE(PY,PZ)-(NY,NZ)
290     PY=NY:PZ=NZ
300     NEXT
310 NEXT
320 PUT(0,0),M,OR
400 LINE(0,0)-(319,199),3,B
410 W$=INPUT$(1):END
1000 'Function
1010 TX=X*2:D=SQR(TX*TX+Y*Y)
1020 IF D<30 OR D>80
        THEN
            Z=0
        ELSE
            Z=30-ABS(55-D)
1030 RETURN

```

PROGRAMME 11.10

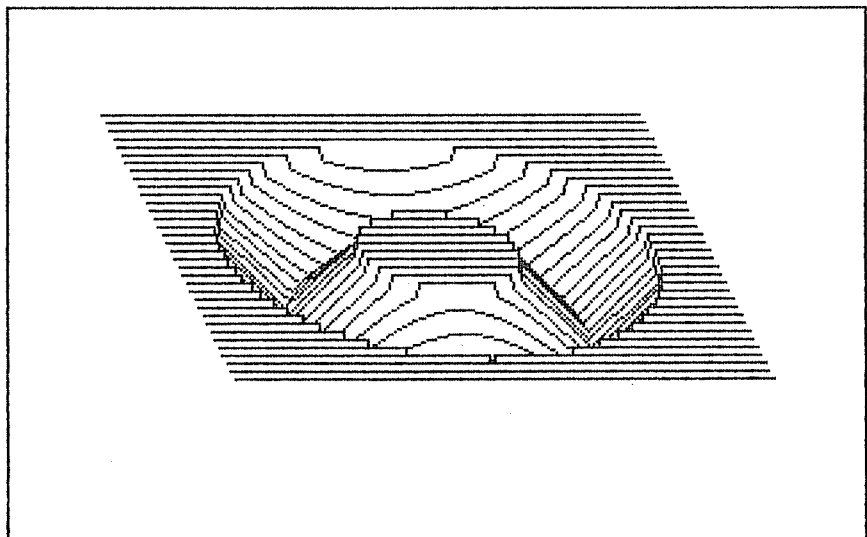


Fig. 11.16

Les lignes 230 à 310 tracent la seconde partie pour différentes valeurs de Y. La figure 11.17 montre l'image résultante. La ligne 320 combine les deux pour produire le graphique que l'on voit figure 11.18.

11.1.8 Quelques fonctions intéressantes

Nous vous présentons ici quelques fonctions intéressantes. En travaillant sur celles-ci et celles étudiées jusqu'à présent, il sera possible de créer

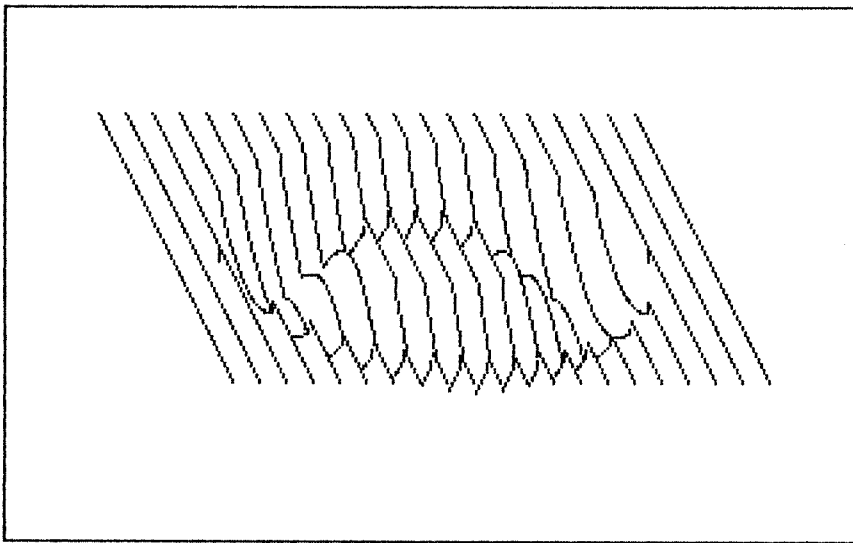


Fig. 11.17

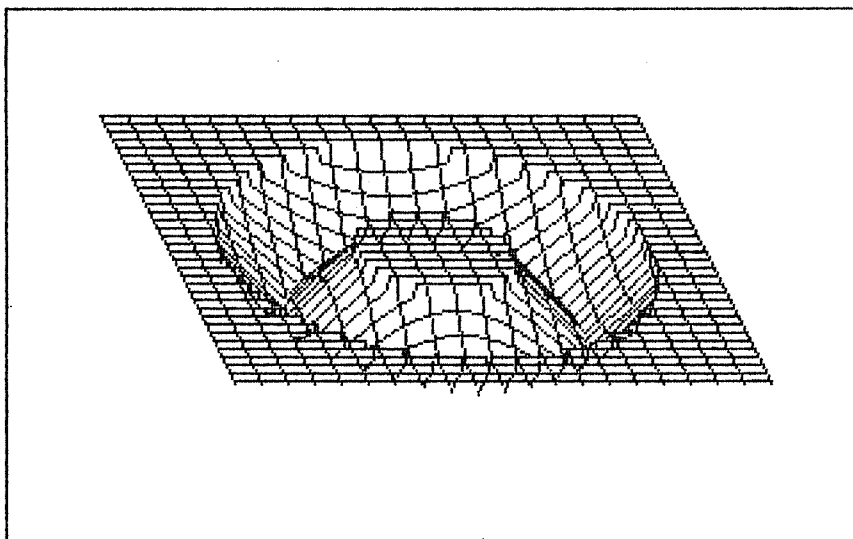


Fig. 11.18

une grande variété de graphiques. Certaines figures ont été réalisées avec le plan $X-Y$ légèrement incliné sur la gauche contrairement à l'inclinaison sur la droite utilisée dans les cas précédents. Ceci est obtenu en soustrayant simplement X (ou une fraction de X) de la position horizontale au lieu de l'y ajouter.

La maison

```

0  '** 11-11 : House **
10 SCREEN 1:COLOR 0,0:CLS
20 FOR X=-80 TO 80 STEP 8
30   FOR Y=-100 TO 100
40     GOSUB 1000
50     NY=Y+160+X*.5:NZ=160-Z+X*.5
60     LINE(NY,NZ)-(NY,199),0
70     IF Y>-100
        THEN
          LINE(PY,PZ)-(NY,NZ)
80     PY=NY:PZ=NZ
90   NEXT
100 NEXT
110 W$=INPUT$(1):END
1000 'Function
1010 IF X<-40 OR X>40
        THEN
          1040
1020 IF Y<-40 OR Y>40
        THEN
          1040
1030 Z=90-ABS(Y):RETURN
1040 Z=20:RETURN

```

PROGRAMME 11.11

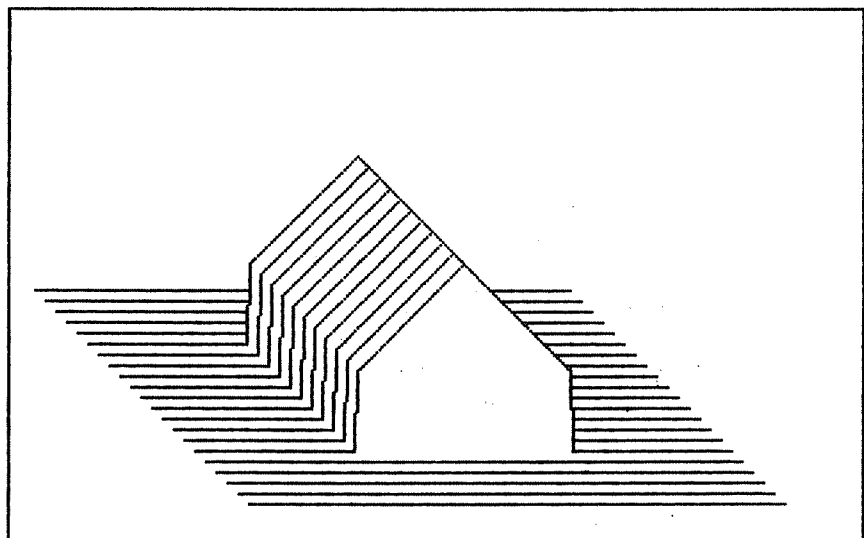


Fig. 11.19

L'anneau

```

0  ** 11-12 : Doughnut **
10 SCREEN 1:CLS
20 FOR X=-80 TO 70 STEP 3
30   FOR Y=-100 TO 100 STEP 10
40     GOSUB 1000
50     NY=Y-X*.7+150:NZ=Z+X+0
60     LINE(NY,NZ)-(NY,199),0
70     IF Y=-100
        THEN
          PSET(NY,NZ)
        ELSE
          LINE(PY,PZ)-(NY,NZ)
80     PY=NY:PZ=NZ
90   NEXT
100 NEXT
110 W$=INPUT$(1)
120 END
1000 'Function
1010 YT=Y*.7
1020 D=SQR(X*X+YT*YT)
1030 IF D<30 OR D>60
    THEN
      Z=100:RETURN
1040 D1=ABS(45-D):Z=100-1.5*SQR(225-D1*D1)
1050 RETURN

```

PROGRAMME 11.12

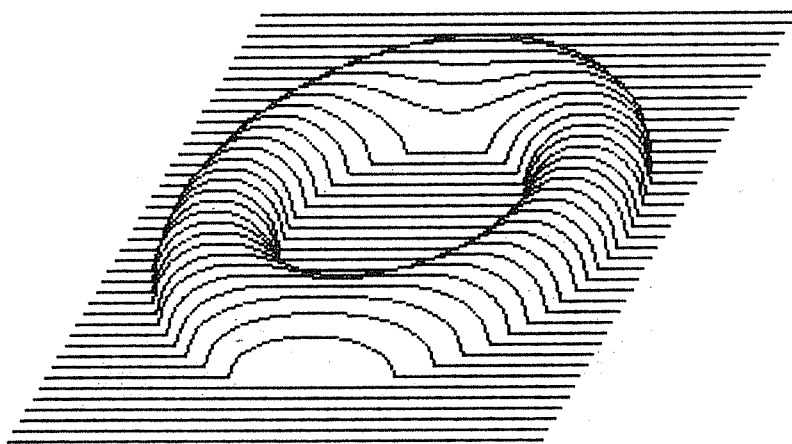


Fig. 11.20

L'anneau avec un plan incliné

```

0  ' ** 11-13 : Doughnut, incl. plane **
10 SCREEN 1:CLS
20 FOR X=-80 TO 70 STEP 3
30   FOR Y=-100 TO 100
40     GOSUB 1000
50     NY=Y-X*.7+150:NZ=Z+X+0
60     LINE(NY,NZ)-(NY,199),0
70     IF Y=-100
        THEN
            PSET(NY,NZ)
        ELSE
            LINE(PY,PZ)-(NY,NZ)
80     PY=NY:PZ=NZ
90   NEXT
100 NEXT
110 W$=INPUT$(1)
120 END
1000 'Function
1010 YT=Y*.7
1020 D=SQR(X*X+YT*YT)
1030 IF D<30 OR D>60
        THEN
            Z=100+Y:RETURN
1040 D1=ABS(45-D):Z=100-1.5*SQR(225-D1*D1)
1050 RETURN

```

PROGRAMME 11.13

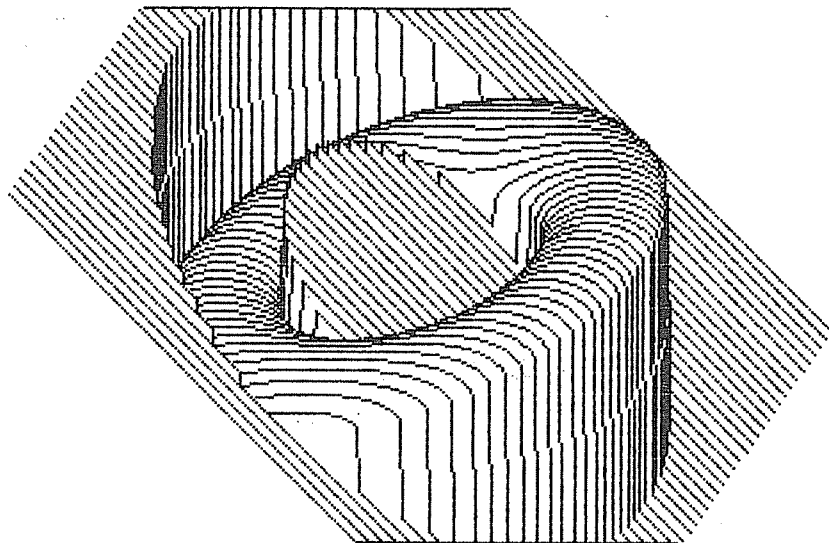


Fig. 11.21

La vague

```
0 '** 11-14 : Wave **
10 SCREEN 1:CLS
20 FOR X=-80 TO 70 STEP 4
30   FOR Y=-100 TO 100
40     GOSUB 1000
50     NY=Y-X*.5+150:NZ=Z+X*.7+120
60     LINE(NY,NZ)-(NY,199),0
70     IF Y=-100
        THEN
          PSET(NY,NZ)
        ELSE
          LINE(PY,PZ)-(NY,NZ)
80   PY=NY:PZ=NZ
90   NEXT
100 NEXT
110 W$=INPUT$(1)
120 END
1000 'Function
1010 YT=Y*.7
1020 D=SQR(X*X+YT*YT)
1030 Z=3*COS(D*.35)
1040 RETURN
```

PROGRAMME 11.14

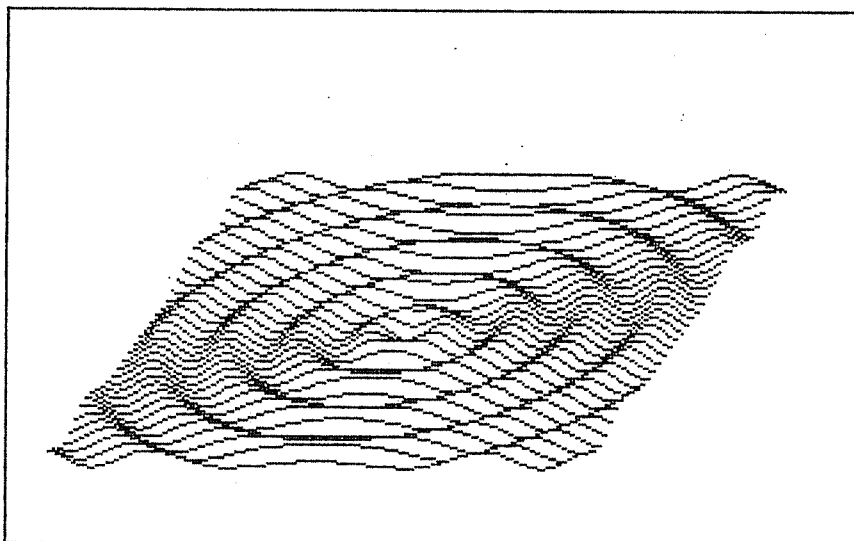


Fig. 11.22

La montagne jusqu'au ciel

```

0  '** 11-15 : Mountain to heaven **
10 SCREEN 1:CLS
20 FOR X=-81 TO 70 STEP 6
30   FOR Y=-100 TO 100
40     GOSUB 1000
50     NY=Y-X*.5+150:NZ=Z+X*.5+60
60     LINE(NY,NZ)-(NY,199),0
70     IF Y=-100
        THEN
          PSET(NY,NZ)
        ELSE
          LINE(PY,PZ)-(NY,NZ)
80     PY=NY:PZ=NZ
90   NEXT
100 NEXT
110 W$=INPUT$(1)
120 END
1000 'Function
1010 XT=X*5.000001E-03:YT=Y*5.000001E-03
1020 Z=100-6/SQR(XT*XT+YT*YT)
1030 RETURN

```

PROGRAMME 11.15

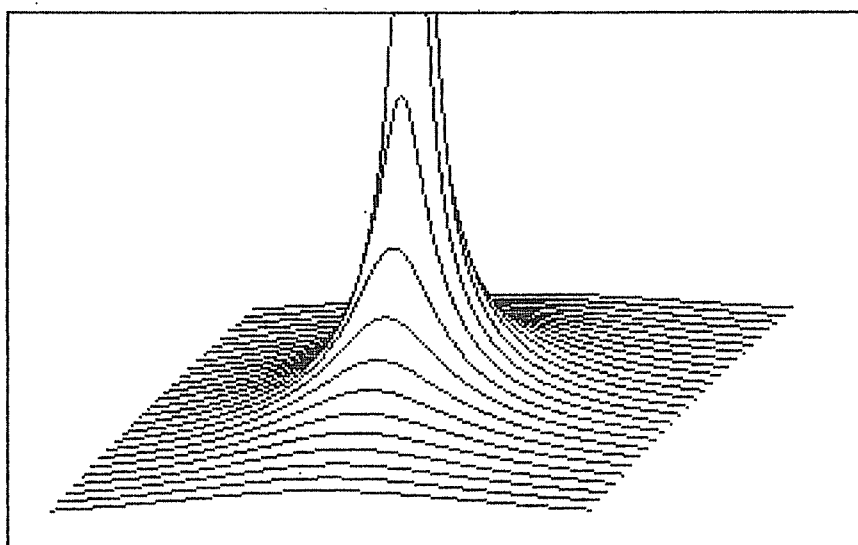


Fig. 11.23

La lettre A

```

0  '** 11-16 : Letter A **
10 SCREEN 1:CLS
15 FA=3.333
20 FOR X=-20 TO 130 STEP 5
30   FOR Y=0 TO 200
40     GOSUB 1000
50     NY=Y-X*.5+80:NZ=Z+X*.6+60
60     LINE(NY,NZ)-(NY,199),0
70     IF Y=0
        THEN
            PSET(NY,NZ)
        ELSE
            LINE(PY,PZ)-(NY,NZ)
80     PY=NY:PZ=NZ
90   NEXT Y
100 NEXT X
110 W$=INPUT$(1)
120 END
1000 'Function
1010 Z=Y*.1:XT=X*.1:YT=(Y+120)*.06
1020 IF XT<0 OR XT>10
        THEN
            1050
1030 IF XT<-FA*(YT-10)+10 OR
        XT<FA*(YT-10)-16.66666
        THEN
            1050
1040 IF XT<-FA*(YT-10)+16.66666 OR
        XT<FA*(YT-10)-10 OR (XT>6 AND XT<8)
        THEN
            Z=-20
1050 RETURN

```

PROGRAMME 11.16

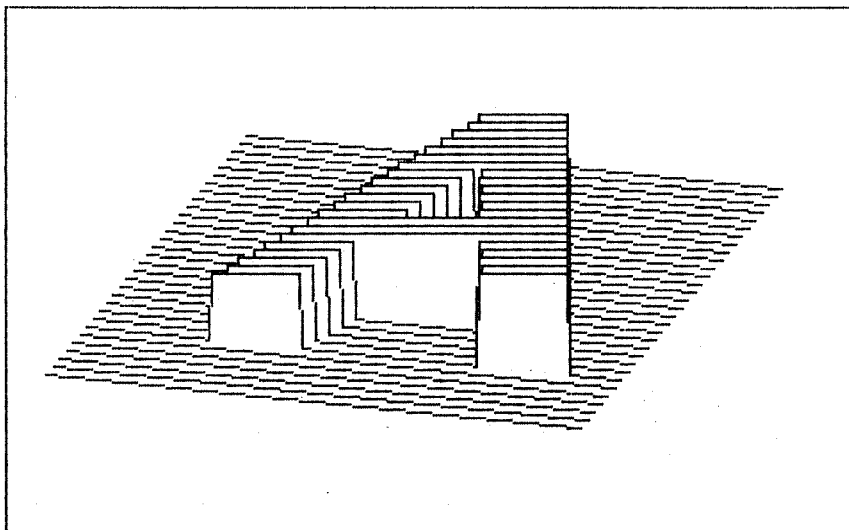


Fig. 11.24

Le cône

```

0  '** 11-17 : Cone **
10 SCREEN 1:CLS
20 FOR X=-100 TO 100 STEP 5
30   FOR Y=-100 TO 100
40     GOSUB 1000
50     NY=Y-X*.5+170:NZ=Z+X*.6+90
60     LINE(NY,NZ)-(NY,199),0
70     IF Y=-100
        THEN
          PSET(NY,NZ)
        ELSE
          LINE(PY,PZ)-(NY,NZ)
80     PY=NY:PZ=NZ
90   NEXT Y
100 NEXT X
110 W$=INPUT$(1)
120 END
1000 'Function
1010 Z=0
1020 D=SQR(X*X+Y*Y)
1030 IF D<90
    THEN
      Z=D-90
1040 RETURN

```

PROGRAMME 11.17

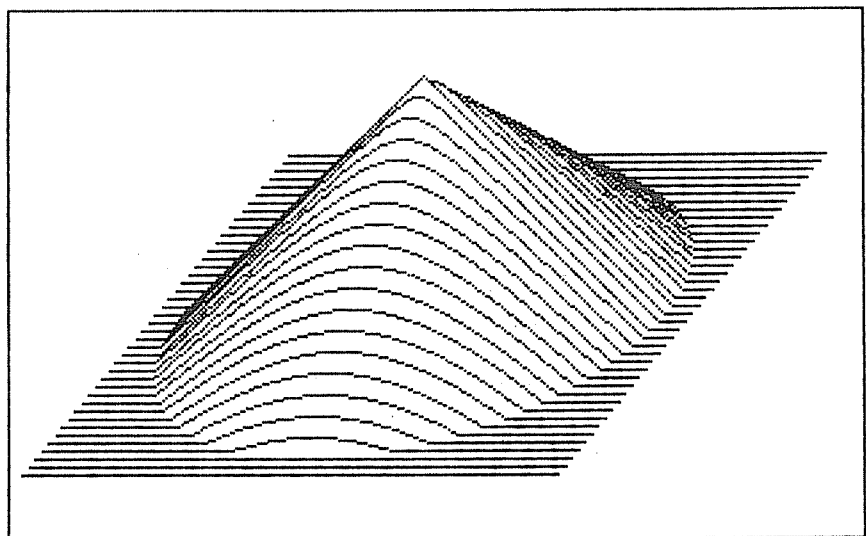


Fig. 11.25

La pyramide

```
0  ** 11-18 : Pyramid **
10 SCREEN 1:CLS
20 FOR X=-100 TO 100 STEP 5
30   FOR Y=-100 TO 100
40     GOSUB 1000
50     NY=Y-X*.5+170:NZ=Z+X*.6+125
60     LINE(NY,NZ)-(NY,199),0
70     IF Y=-100
        THEN
          PSET(NY,NZ)
        ELSE
          LINE(PY,PZ)-(NY,NZ)
80     PY=NY:PZ=NZ
90   NEXT Y
100 NEXT X
110 W$=INPUT$(1)
120 END
1000 'Function
1010 Z=Y*.1
1020 D=SQR(X*X+Y*Y)
1030 IF D>90
    THEN
      RETURN
1040 Z=10*INT(D/10)-90
1050 RETURN
```

PROGRAMME 11.18

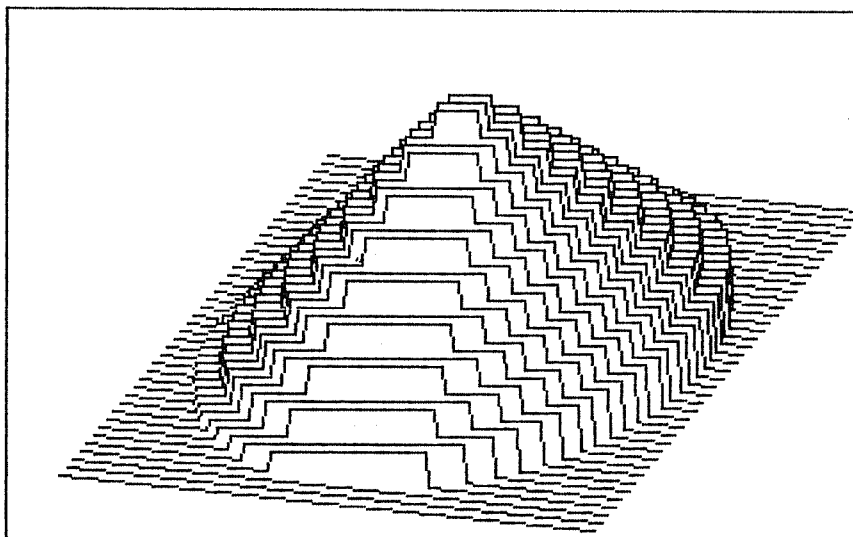


Fig. 11.26

Le château

```

0  '** 11-19 : Lovecraft's castle **'
10 SCREEN 1:CLS
20 FOR X=-100 TO 100 STEP 5
30   FOR Y=-100 TO 100
40     GOSUB 1000
50     NY=Y-X*.5+170:NZ=Z+X*.6+125
60     LINE(NY,NZ)-(NY,199),0
70     IF Y=-100
        THEN
            PSET(NY,NZ)
        ELSE
            LINE(PY,PZ)-(NY,NZ)
80     PY=NY:PZ=NZ
90   NEXT Y
100 NEXT X
110 W$=INPUT$(1)
120 END
1000 'Function
1010 XT=20*(X\20):YT=10*(Y\10)
1020 Z=Y*.2
1030 D=SQR(XT*XT+YT*YT)
1040 D=18*INT(D\18)
1050 IF D>80
        THEN
            RETURN
1060 ZT=D-80:
        IF ZT<Z
            THEN
                Z=ZT
1070 RETURN

```

PROGRAMME 11.19

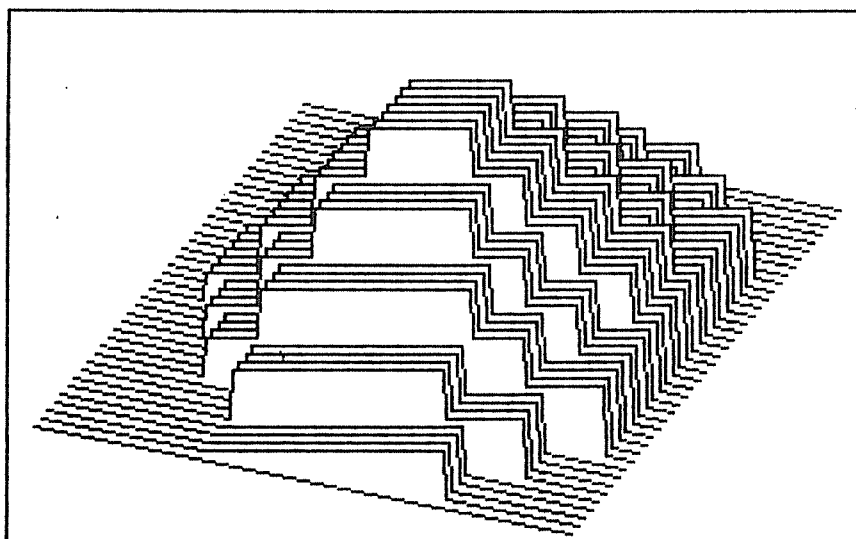


Fig. 11.27

La croix

```

0  '** 11-20 : Cross **
10 SCREEN 1:CLS
20 FOR X=-100 TO 100 STEP 5
30   FOR Y=-100 TO 100 STEP 1
40     GOSUB 1000
50     NY=Y+160-X*.6:NZ=Z+X*.5
60     LINE(NY,NZ)-(NY,199),0
70     IF Y>-100
        THEN
          LINE(PY,PZ)-(NY,NZ)
80     PY=NY:PZ=NZ
90   NEXT
100 NEXT
110 W$=INPUT$(1)
120 END
1000 'Function
1010 IF X<-80 OR X>80
    THEN
      1050
1020 IF Y>-30 AND Y<30
    THEN
      Z=85:RETURN
1030 IF Y<-80 OR Y>80
    THEN
      1050
1040 IF X>-30 AND X<30
    THEN
      Z=85:RETURN
1050 Z=100:RETURN

```

PROGRAMME 11.20

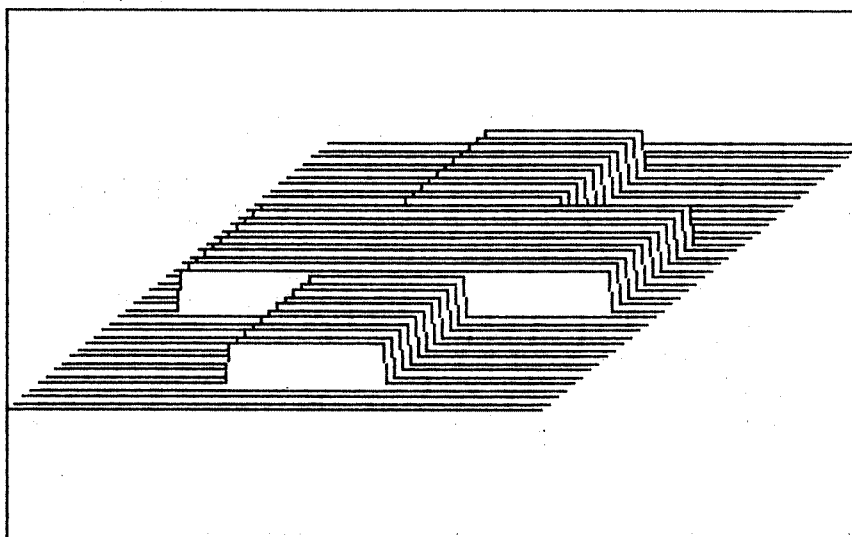


Fig. 11.28

La croix avec un plan incliné

```

0 '** 11-21 : Cross, inclined plane *
10 SCREEN 1:CLS
20 FOR X=-100 TO 100 STEP 5
30   FOR Y=-100 TO 100 STEP 1
40     GOSUB 1000
50     NY=Y+160-X*.6:NZ=Z+X*.5
60     LINE(NY,NZ)-(NY,199),0
70     IF Y>-100
        THEN
            LINE(PY,PZ)-(NY,NZ)
80     PY=NY:PZ=NZ
90   NEXT
100 NEXT
110 W$=INPUT$(1)
120 END
1000 'Function
1010 IF X<-80 OR X>80
    THEN
        1050
1020 IF Y>-30 AND Y<30
    THEN
        Z=85:RETURN
1030 IF Y<-80 OR Y>80
    THEN
        1050
1040 IF X>-30 AND X<30
    THEN
        Z=85:RETURN
1050 Z=100+Y*.1:RETURN

```

PROGRAMME 11.21

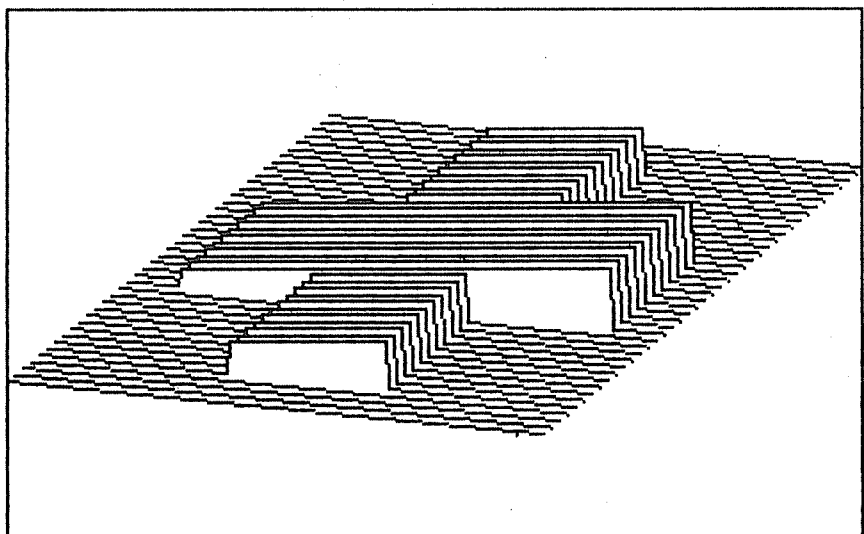


Fig. 11.29

Rotation des fonctions sur deux dimensions

En utilisant les équations de rotation 2-D expliquées au paragraphe 8.2.3, nous pouvons changer facilement l'orientation de la fonction. Si les coordonnées de chaque point (X,Y) sont modifiées par les équations

$$X1 = \cos(\text{PHI}) * X + \sin(\text{PHI}) * Y$$

$$Y1 = -\sin(\text{PHI}) * X + \cos(\text{PHI}) * Y$$

et qu'on utilise les valeurs résultantes X1 et Y1 pour calculer la fonction, le graphique résultant sera la représentation de la figure d'origine ayant tourné d'un angle PHI (en radians). Le programme 11.22 trace la croix de la figure 11.29, mais tournée de 45 degrés, comme on peut voir figure 11.30.

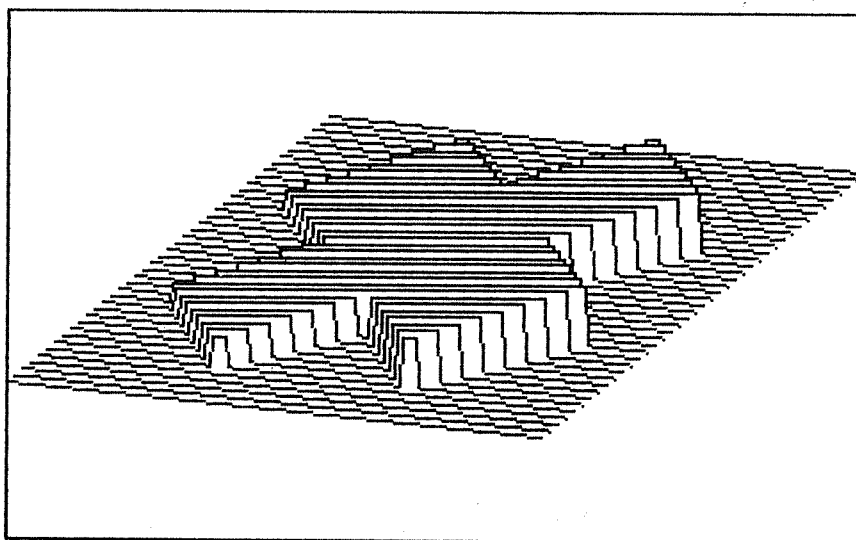


Fig. 11.30

```
0  ** 11-22 : Cross with rotation **
10 CLS: INPUT "angle "; AN: AN=AN*1.745329E-02
20 SN=SIN(AN): CS=COS(AN)
30 SCREEN 1:CLS
40 FOR X=-100 TO 100 STEP 5
50   FOR Y=-100 TO 100
60     GOSUB 1000
70     NY=Y+160-X*.6: NZ=Z+X*.5
80     LINE(NY,NZ)-(NY,199),0
90     IF Y>-100
       THEN
         LINE(PY,PZ)-(NY,NZ)
```

```

100     PY=NY:PZ=NZ
110     NEXT
120 NEXT
130 W$=INPUT$(1)
140 END
1000 'Function
1010 XC=CS*X+SN*Y:YC=-SN*X+CS*Y
1020 IF XC<-80 OR XC>80
      THEN
        1060
1030 IF YC>-30 AND YC<30
      THEN
        Z=85:RETURN
1040 IF YC<-80 OR YC>80
      THEN
        1060
1050 IF XC>-30 AND XC<30
      THEN
        Z=85:RETURN
1060 Z=100+Y*.1:RETURN

```

PROGRAMME 11.22

11.1.10 La perspective

Toutes les fonctions 3-D étudiées jusqu'ici ont été tracées sans perspective, c'est-à-dire comme si la taille apparente des parties éloignées était la même que celles qui sont plus proches. Dans la réalité, les objets ont l'air plus petits quand leur distance à l'observateur croît, l'exemple typique étant les rails d'un train qui donnent l'impression de converger vers un point à l'infini. Ce point est en général appelé le point de fuite ², et chaque ligne ayant la même orientation converge vers lui. La figure 11.31 montre un cube et une maison dessinés en perspective.

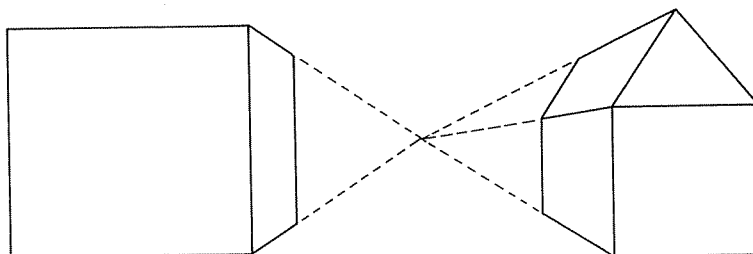


Fig. 11.31

Nous allons rajouter la perspective au plan X-Y de nos fonctions. Pour faciliter le processus, nous tracerons la valeur Y=0 exactement au milieu de l'écran (à la colonne 160), les valeurs négatives sur la gauche et les valeurs positives sur la droite. Notre dessein sera de trouver un

facteur approprié pour chaque point. Au début ce facteur sera petit et donc les distances seront raccourcies. Quand le facteur croîtra, la représentation des distances fera de même. Supposez que nous décidions de commencer avec $FACTOR=0$; toutes les valeurs prises par Y sont multipliées par zéro et deviennent donc un simple point, le point à l'infini. Si nous incrémentons le facteur par 0,1, par exemple, chaque distance sera réduite au 1/10ème de sa taille d'origine, et la ligne tracée sera très petite, comme si on la voyait de très loin. Quand le facteur deviendra égal à 1, après plusieurs incrémentations successives, chaque distance sera tracée à sa taille normale, et si le facteur devient supérieur à 1, la ligne sera agrandie. Le programme 11.23 trace la grille avec la fonction $F(X,Y)=0$, c'est-à-dire le plan X-Y.

```

10 ** 11-23 : Perspective grid **
20 SCREEN 1:CLS
22 MAX.X=100:VANISH.PT=-100
25 DIST=MAX.X-VANISH.PT
30 FOR X=-100 TO 100 STEP 5
35   FACTOR=(X-VANISH.PT)/DIST
40   FOR Y=-100 TO 100 STEP 1
50     GOSUB 1000
70     NY=Y*FACTOR+160:
       IF CINT(NY)=PY
           THEN
               110
75     NZ=Z+X*.5+100
80     LINE(NY,NZ)-(NY,199),0
90     IF Y=-100
           THEN
               PSET(NY,NZ)
           ELSE
               LINE(PY,PZ)-(NY,NZ)
100    PY=CINT(NY):PZ=NZ
110  NEXT
120 NEXT
130 FOR Y=-100 TO 100 STEP 10
140   FOR X=-100 TO 100 STEP 2
145     FACTOR=(X-VANISH.PT)/DIST
150     GOSUB 1000
170     NY=Y*FACTOR+160:
       IF CINT(NY)=PY
           THEN
               210
175     NZ=Z+X*.5+100
190     IF X=-100
           THEN
               PSET(NY,NZ)
           ELSE
               LINE(PY,PZ)-(NY,NZ)
200    PY=NY:PZ=NZ
210  NEXT
220 NEXT
230 W$=INPUT$(1)
240 END

```

```
1000 'Function  
1010 Z=0:RETURN
```

PROGRAMME 11.23

La figure 11.32 montre comment les lignes convergent vers l'infini. Le facteur de perspective doit être tel que :

FACTOR = 0 si le point est le point de fuite

FACTOR = 1 si le point est le plus proche de l'observateur

et tout ce qui se trouve entre les deux est adapté en conséquence. La première chose que nous faisons est de déterminer la distance entre le point de fuite et le point le plus proche de l'observateur, qui est la valeur maximum prise par X. Le point de fuite est purement arbitraire, mais devra être inférieur au plus petit X. La distance est calculée avec l'équation :

$$\text{DISTANCE} = \text{MAXIMUM.X} - \text{VANISHING.POINT.}$$

Nous déterminons ensuite FACTOR pour chaque X au moyen de l'équation

$$\text{FACTOR} = (\text{X} - \text{VANISHING.POINT}) / \text{DISTANCE.}$$

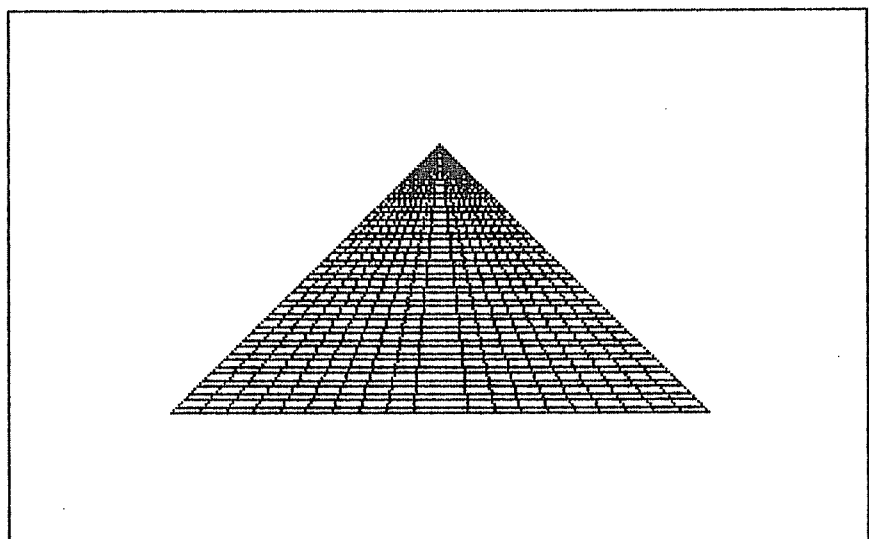


Fig. 11.32

Exemple 1.

Supposez que X varie entre -100 et + 100 et que le point de fuite soit -100. Nous déterminons d'abord la distance prise par X :

$$\text{DISTANCE} = 100 - (-100) = 200.$$

Le facteur pour X = -100, le point le plus éloigné de la fonction est :

$$\text{FACTOR} = (-100 - (-100)) / 200 = 0,$$

ce qui signifie que tous les points pour cette valeur de X seront tracés au centre de l'écran. Le facteur quand X = 100 est :

$$\text{FACTOR} = (100 - (-100)) / 200 = 1,$$

ce qui signifie que tous les points seront affichés comme ils sont. Finalement, voyons ce qui se passe au milieu. Le facteur pour X = 0 est :

$$\text{FACTOR} = (0 - (-100)) / 200 = 0.5,$$

ce qui signifie qu'ici l'échelle de la courbe sera exactement la moitié de ce qu'elle aurait été sans perspective.

Exemple 2.

Supposez que X varie de -100 à 100 et que le point de fuite soit à -300. La distance est :

$$\text{DISTANCE} = 100 - (-300) = 400.$$

Quand X = 100, le facteur est :

$$\text{FACTOR} = (-100 - (-300)) / 400 = 0.5,$$

ce qui signifie que l'échelle de la courbe sera la moitié de ce qu'elle aurait été sans perspective. Quand X = 0,

$$\text{FACTOR} = (0 - (-300)) / 400 = 0.75,$$

qui réduira la courbe aux trois quarts de sa taille normale. Finalement, quand X = 100,

$$\text{FACTOR} = (100 - (-300)) / 400 = 1$$

qui laisse la courbe telle quelle.

Notez que les dimensions verticale et horizontale sont toutes les deux affectées par la perspective en profondeur. Par exemple, dans le cube de la figure 11.31, le carré du fond est plus petit que celui de devant ; ce qui signifie que sa largeur et sa hauteur ont été toutes les deux modifiées par la perspective. Cette double variation peut être réalisée si non seulement la largeur (coordonnée Y) est multipliée par le facteur de perspective, mais aussi la hauteur (coordonnée Z).

Le programme 11.24 trace la fameuse fonction $F(X,Y) = \sin(X) * \sin(Y)$ en perspective. A la ligne 1010, Z est multiplié par le facteur de perspective pour la hauteur calculé en ligne 50. A la ligne 80, Y est également multiplié par le facteur de perspective. Comme le pas (STEP) est 1 la distance entre les Y successifs sera pour la plupart du temps inférieure à 1, ceci ayant pour résultat plusieurs points positionnés sur la même colonne. Pour éviter ceci, on compare NY (le Y affecté du facteur de perspective) avec le précédent Y. S'ils sont égaux, NY n'est pas tracé. Comme les valeurs fractionnaires sont arrondies avant d'être tracées sur l'écran, NY et PY sont aussi arrondis à l'entier le plus proche avec la fonction CINT. La figure 11.33 montre cette vue en perspective.

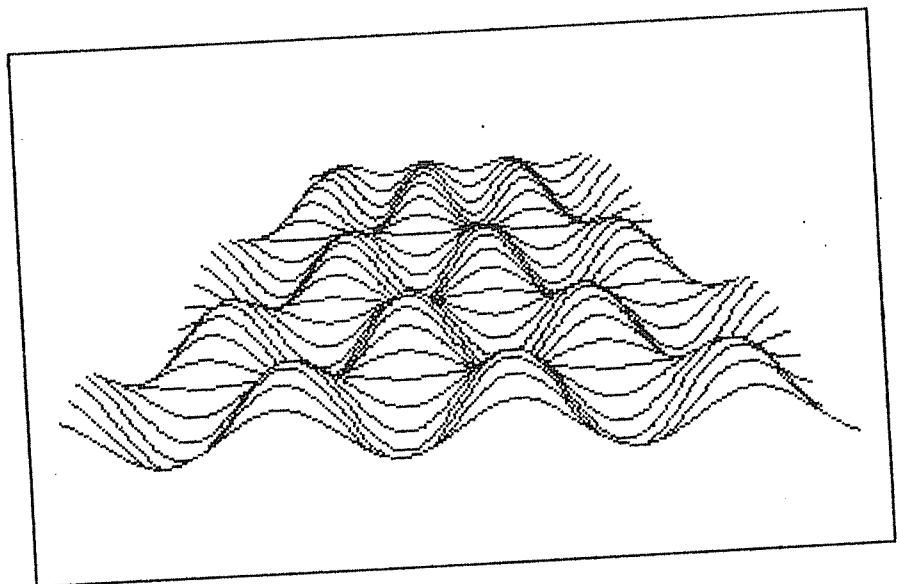


Fig. 11.33

```
0  '** 11-24 : Sine with perspective **
10 SCREEN 1:CLS
20 MAX.X=100:INPUT"VANISH.PT ";VANISH.PT:CLS
30 DIST=MAX.X-VANISH.PT
40 FOR X=-100 TO 100 STEP 7
50   FACTOR=(X-VANISH.PT)/DIST
60   FOR Y=-150 TO 150
```



```

70   GOSUB 1000
80   NY=CINT(Y*FACTOR+160):
    IF NY=PY
    THEN
    130
90   NZ=Z+X*.5+100
100  LINE(NY,NZ)-(NY,199),0
110  IF Y=-150
    THEN
    PSET(NY,NZ)
    ELSE
    LINE(PY,PZ)-(NY,NZ)
120  PY=NY:PZ=NZ
130  NEXT
140 NEXT
150 W$=INPUT$(1)
160 END
170 W$=INPUT$(1)
180 END
1000 'Function
1010 Z=20*SIN(X*.06)*SIN(Y*.07)*FACTOR
1020 RETURN

```

PROGRAMME 11.24

Le programme 11.25 dessine un anneau sur un plan incliné et en perspective. La figure 11.34 montre le graphique produit avec un point de fuite à -300 et la figure 11.35 montre que la perspective avec un point de fuite à -1000 devient très discrète, presque non discernable.

```

0   '** 11-25 Doughnut in perspective **
10  SCREEN 1:CLS
20  MAX.X=100:INPUT"VANISH.PT ";VANISH.PT:CLS
30  DIST=MAX.X-VANISH.PT
40  FOR X=-100 TO 100 STEP 7
50    FACTOR=(X-VANISH.PT)/DIST
60    FOR Y=-150 TO 150
70      GOSUB 1000
80      NY=CINT(Y*FACTOR+160):
    IF NY=PY
    THEN
    130
90    NZ=Z+X*.5+100
100   LINE(NY,NZ)-(NY,199),0
110   IF Y=-150
    THEN
    PSET(NY,NZ)
    ELSE
    LINE(PY,PZ)-(NY,NZ)
120   PY=NY:PZ=NZ
130   NEXT
140 NEXT
150 W$=INPUT$(1)
160 END
170 W$=INPUT$(1)
180 END
1000 YT=Y*.6:XT=X*.7

```

```

1010 D=SQR(XT*XT+YT*YT)
1020 IF D<30
    THEN
        Z=150:RETURN
1025 IF D<30 OR D>60
    THEN
        Z=0-Y*.1+X*.3:RETURN
1030 D1=ABS(45-D):
    Z=0-1.5*FACTOR*SQR(225-D1*D1)
1040 RETURN

```

PROGRAMME 11.25

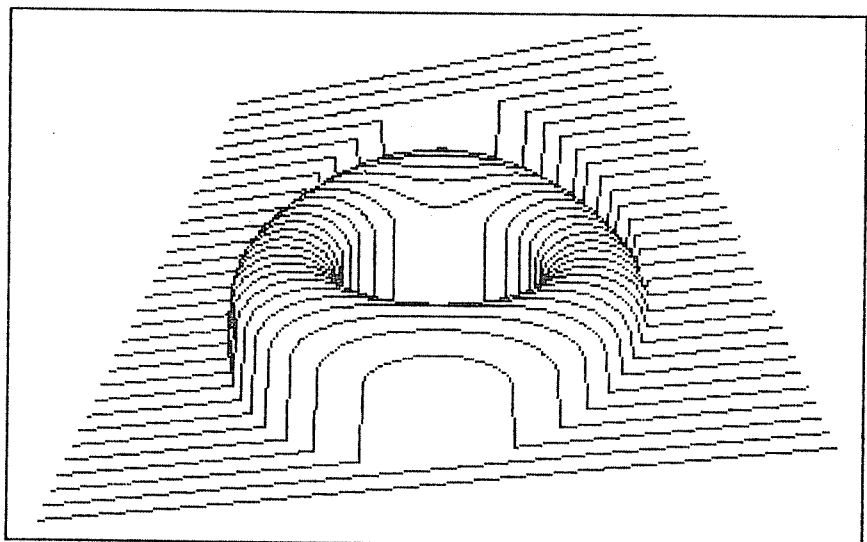


Fig. 11.34

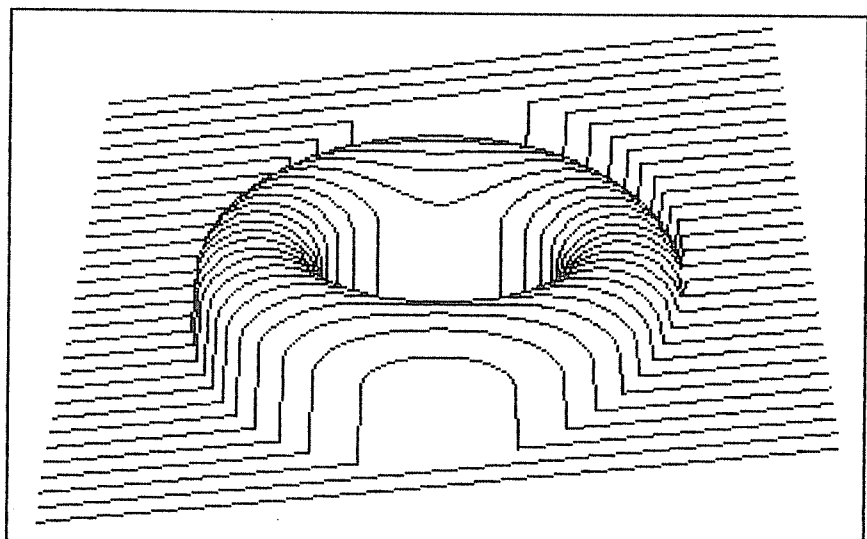


Fig. 11.35

Le point de fuite ne devra jamais être plus grand que la plus petite valeur prise par X , ou sinon la fonction sera complètement déformée. Des points de fuite plus grands que le X maximum produisent une inversion de la perspective, c'est-à-dire que le plan arrière sera plus grand que le plan avant.

11.2 OBJETS TRIDIMENSIONNELS

Des images tridimensionnelles qui se déplacent et tournent dans l'espace apparaissent constamment à la télévision et dans les films, et beaucoup de branches de la science et de la technique utilisent ces images pour simuler et étudier des objets dont il serait par ailleurs difficile et coûteux de réaliser des modèles réels. La conception assistée par ordinateur (CAO) est un terrain florissant qui profite énormément de ces représentations d'objets tridimensionnels. Pour travailler avec des objets solides, nous devons avoir recours au système de coordonnées 3-D présenté au début de ce chapitre.

11.2.1 Les projections

Comme l'écran et les feuilles de papier sur lesquelles l'imprimante peut recopier l'écran, sont tous les deux plats, il est nécessaire de trouver un moyen de représenter ces objets tridimensionnels avec seulement deux dimensions. Une de ces méthodes est la projection : par exemple dans une pièce éclairée avec une seule lampe, les ombres produites par des objets solides (une chaise, une personne, etc...) sur les murs sont des représentations bidimensionnelles d'objets tridimensionnels.

Toutes les formes manipulées dans les chapitres précédents peuvent être considérées comme tridimensionnelles, avec une coordonnée Z nulle. Par exemple un cercle de rayon 80 centré en $(160,100)$ est un ensemble de points décrits par un triplet (X,Y,Z) au lieu d'un couple (X,Y) et toutes les coordonnées Z sont fixées à zéro, ainsi le cercle se trouvera sur le plan $X-Y$ et sera identique à un cercle 2-D. Si par exemple nous fixons la composante Z égale à 10, le cercle se trouvera 10 unités au-dessus du plan $X-Y$ comme le montre la figure 11.36.

Si nous considérons le plan $X-Y$ comme parallèle au sol et regardons le cercle de la figure 11.36 du dessus, c'est-à-dire avec l'œil centré sur l'axe des Z , nous ne pourrions pas dire si le cercle se trouve sur le plan $X-Y$, au-dessus ou en dessous³. Cet exemple montre le type de projections que nous utiliserons pour aplanir des objets tridimensionnels : nous ignorerons simplement la composante Z de chaque point au moment de l'affichage. Les composantes Z sont vitales quand on calcule la transformation d'objets, comme les rotations ou les déformations en perspective, mais quand nous montrons des objets sur l'écran nous ne les

utiliserons plus. La figure 11.37 montre la projection d'un cube sur le plan X-Y.

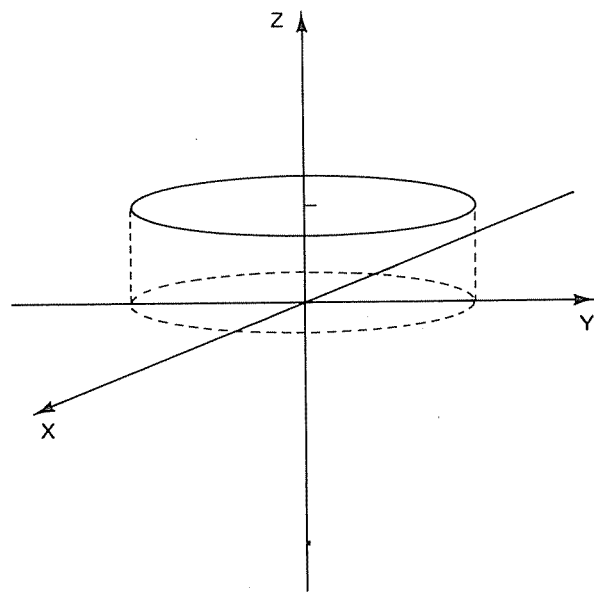


Fig. 11.36

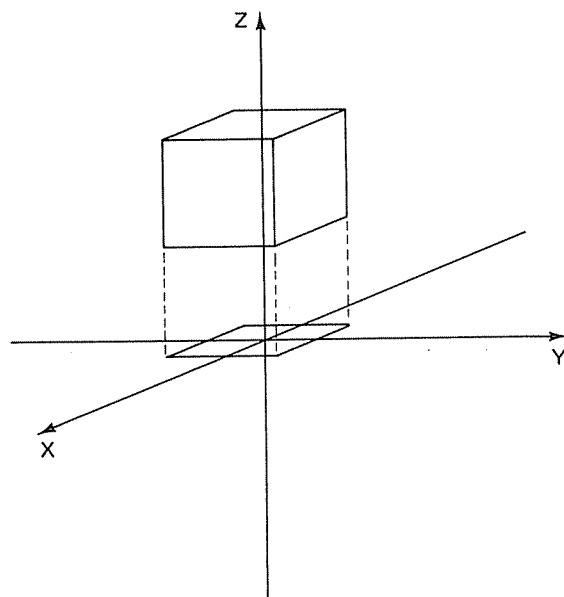


Fig. 11.37

11.2.2 Représentation interne

Nous utiliserons deux types de représentations différentes pour les objets solides : la représentation vectorielle et la représentation par algorithme.

Représentation vectorielle. Quand un objet comporte un petit nombre de points, nous utiliserons la représentation vectorielle présentée au paragraphe 2.7 : on utilisera trois tableaux (X, Y et Z) pour mémoriser les coordonnées des points limites de chaque vecteur de l'objet, et deux tableaux (LO et LD) pour mémoriser l'origine et la destination des lignes qui relient ces points. Par exemple, le cube que l'on voit à la figure 11.38 sera représenté intérieurement par les points 1, 2, 3, 4, 5, 6, 7 et 8 et par les lignes 1-2 (qui indique que le point 1 est relié au point 2 par une ligne droite), 2-3, 3-4, 4-1, 5-6, 6-7, 7-8, 8-5, 1-5, 2-6, 3-7 et 4-8.

Si par exemple le cube a un côté de longueur 50, les valeurs des tableaux seront celles de la table 11.1. La projection de ce cube sur le plan X-Y est un carré. Le programme 11.26 rentre les points et les vecteurs qui représentent un objet solide. La boucle FOR des lignes 140 à 160 affiche l'objet en traçant les lignes entre les points limites de chaque ligne (de l'origine à la destination) tout en ignorant la composante Z, c'est-à-dire en faisant une projection sur le plan X-Y.

```
0  '** 11-26 : 3-D plot with vectors **
10 SCREEN 1:CLS:
   PRINT"Type 999,0,0 to end":PRINT
20 DIM X(100),Y(100),Z(100),LO(100),LD(100)
30 FOR I=1 TO 100
40   PRINT"Point #";I;"(X,Y,Z) ";
50   INPUT X(I),Y(I),Z(I)
60   IF X(I)=999
      THEN
        NP=I-1:GOTO 80
70 NEXT I:NP=I-1
75 PRINT:PRINT"Type 999,0 to end":PRINT
80 FOR I=1 TO 100
90   PRINT"Line #";I;"(Origin,destination)";
100  INPUT LO(I),LD(I)
110  IF LO(I)=999
      THEN
        NL=I-1:GOTO 130
120 NEXT I:NL=I-1
130 CLS
140 FOR I=1 TO NL
150   LINE(160+X(LO(I)),100+Y(LO(I)))-
      (160+X(LD(I)),100+Y(LD(I)))
160 NEXT
```

PROGRAMME 11.26

Table 11.1

POINT	X	Y	Z
1	0	0	0
2	50	0	0
3	50	50	0
4	0	50	0
5	0	0	50
6	50	0	50
7	50	50	50
8	0	50	50

LIGNE	LO	LD
1	1	2
2	2	3
3	3	4
4	4	1
5	5	6
6	6	7
7	7	8
8	8	5
9	1	5
10	2	6
11	3	7
12	4	8

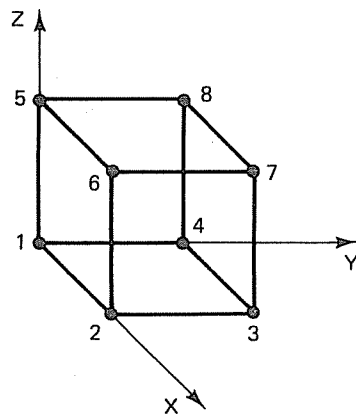


Fig. 11.38

Un cube décrit comme celui décrit à la table 11.1 ne produit pas un graphique très intéressant quand on le projette sur le plan X-Y. Essayez

plutôt le cube tourné dont les points sont donnés à la table 11.2. Les lignes sont celles de la table 11.1.

Table 11.2

POINT	X	Y	Z
1	0	0	0
2	45	9	-19
3	45	54	2
4	0	45	21
5	21	-19	41
6	66	-10	22
7	66	35	43
8	21	26	62

Représentation par algorithme. Quand un objet est généré par un algorithme qui calcule un grand nombre de points, il est plus pratique d'utiliser l'algorithme directement et de soumettre chaque point aux sous-routines de transformation (qu'on étudiera aux paragraphes suivants). Par exemple, un grand cercle calculé en coordonnées polaires occupera une place importante en mémoire si on le mémorise sous forme vectorielle. Si ses lignes sont tracées directement (après les transformations désirées), la mémoire ne sera pas gaspillée et le processus s'exécutera plus vite.

La figure 11.39 montre un cône tracé avec une spirale dont la composante Z croît proportionnellement au rayon. Le programme qui le produit sera expliqué au paragraphe 11.2.8.

11.2.3 Les rotations

Les rotations fournissent une des manipulations les plus intéressantes d'objets tridimensionnels. En faisant tourner les objets autour d'un ou plusieurs des trois axes, il est possible de faire varier la position relative du point d'observation, permettant ainsi l'observation de dessus, de dessous ou de côté, etc. Si par exemple un programme possède l'information nécessaire pour tracer le Golden Gate Bridge (le grand pont de San Francisco), des rotations permettront la simulation d'un vol au-dessus de lui, en dessous, à travers lui et même la vue qu'on aurait si on se tenait au sommet d'un des câbles ou des montants !

En manipulant simplement les trois projections simples du paragraphe 11.2.1 il est possible d'avoir trois points de vue différents de l'objet.

Le programme 11.27 produit le mot TITO en caractères pleins, comme le montre la figure 11.40. La figure 11.41 indique la direction de vision. Le graphique ne met pas bien en évidence l'effet de volume : c'est comme lorsqu'on regarde un cube de face et que la forme obtenue à l'air complètement plate.

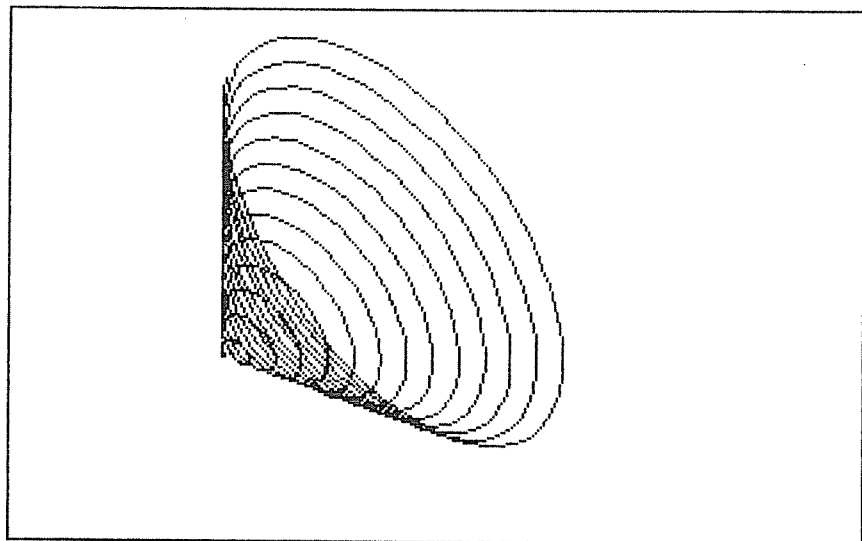


Fig. 11.39

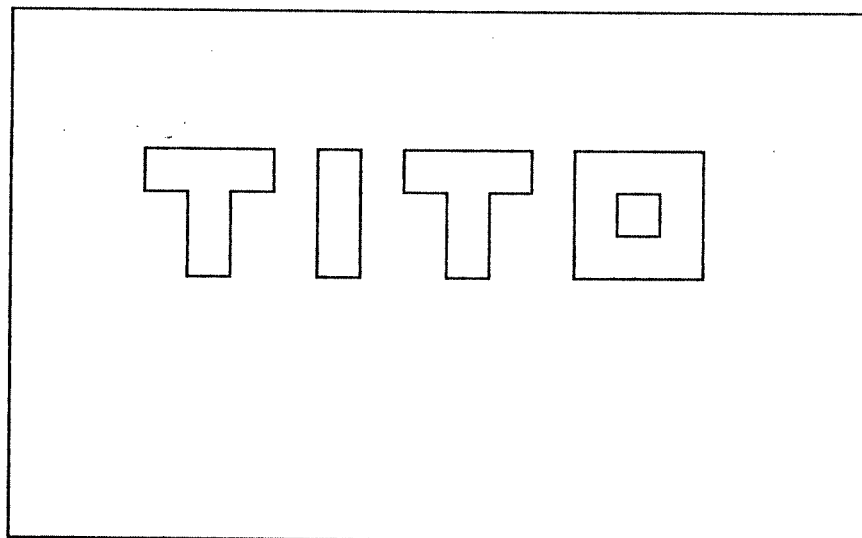


Fig. 11.40

```
0 ' ** 11-27 : Plot "Tito" **
10 SCREEN 1:CLS
20 DIM X(60),Y(60),Z(60),LD(100),LD(100)
30 FOR I=1 TO 56:
    READ X(I),Y(I),Z(I):
```



```

NEXT
40 FOR I=1 TO 84:
  READ LO(I),LD(I):
  NEXT
50 FOR I=1 TO 84
60  LINE(50+X(LD(I)),100+Y(LD(I)))-
    (50+X(LD(I)),100+Y(LD(I)))
70 NEXT:END
5000 DATA 0,-48,0,0,-32,0,16,-32,0,16,0,
    0,32,0,0,32,-32,0,48,-32,0,48,
    -48,0,0,-48,16,0,-32,16
5010 DATA 16,-32,16,16,0,16,32,0,16,32,
    -32,16,48,-32,16,48,-48,16,64,
    -48,0,64,0,0,80,0,0,80,-48,0
5020 DATA 64,-48,16,64,0,16,80,0,16,80,
    -48,16,96,-48,0,96,-32,0,112,
    -32,0,112,0,0,128,0,0,128,-32,0
5030 DATA 144,-32,0,144,-48,0,96,-48,16,
    96,-32,16,112,-32,16,112,0,16,
    128,0,16,128,-32,16,144,-32,16,
    144,-48,16
5040 DATA 160,-48,0,160,0,0,208,0,0,208,-48,
    0,176,-32,0,176,-16,0,192,-16,0,
    192,-32,0,160,-48,16,160,0,16
5050 DATA 208,0,16,208,-48,16,176,-32,16,
    176,-16,16,192,-16,16,192,-32,16
5060 DATA 1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,1
5070 DATA 9,10,10,11,11,12,12,13,13,
    14,14,15,15,16,16,9
5080 DATA 1,9,2,10,3,11,4,12,5,13,6,
    14,7,15,8,16
5090 DATA 17,18,18,19,19,20,20,17
5100 DATA 21,22,22,23,23,24,24,21
5110 DATA 17,21,18,22,19,23,20,24
5120 DATA 25,26,26,27,27,28,28,29,
    29,30,30,31,31,32,32,25
5130 DATA 33,34,34,35,35,36,36,37,
    37,38,38,39,39,40,40,33
5140 DATA 25,33,26,34,27,35,28,36,
    29,37,30,38,31,39,32,40
5150 DATA 41,42,42,43,43,44,44,41,
    49,50,50,51,51,52,52,49
5160 DATA 45,46,46,47,47,48,48,45,
    53,54,54,55,55,56,56,53
5170 DATA 41,49,42,50,43,51,44,52,
    45,53,46,54,47,55,48,56

```

PROGRAMME 11.27

La projection obtenue en prenant les composantes X et Y de chaque point (ligne 60) peut être modifiée si on prend X et Z à la place et qu'on ignore complètement la composante Y. Quand on change la ligne 60 du programme 11.27 en :

```

60 LINE(50+X(LO(I)),100+Z(LD(I)))-
    (50+X(LD(I)),100+Z(LD(I)))

```

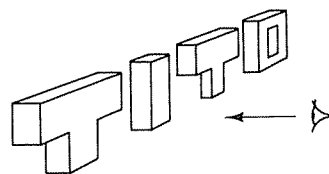


Fig. 11.41

l'objet est vu du dessus, comme le montre la figure 11.42. La figure 11.43 indique la direction de vision.

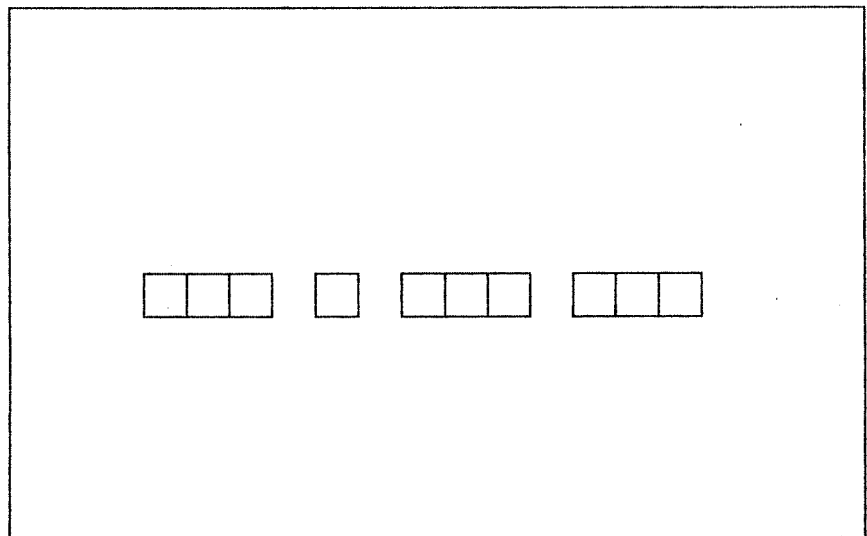


Fig. 11.42

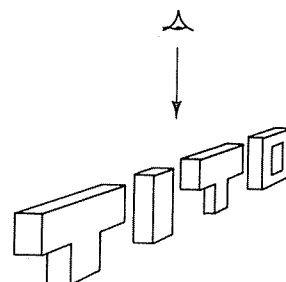


Fig. 11.43

On peut se placer d'une autre point de vue si on ignore la composante X de chaque point et qu'on utilise chaque Y et Z pour tracer les vecteurs. Si la ligne 60 du programme 11.27 est changée en :

```
60 LINE(50+Y(LO(I)),100+Z(LO(I)))-  
      (50+Y(LD(I)),100+Z(LD(I)))
```

l'objet est dessiné d'un autre point de vue, comme le montre la figure 11.44. La figure 11.45 montre la direction de vision.

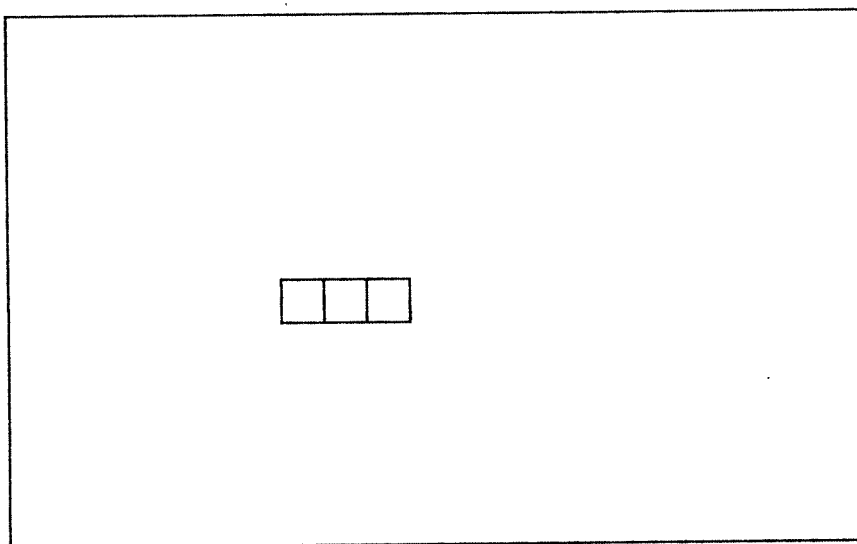


Fig. 11.44

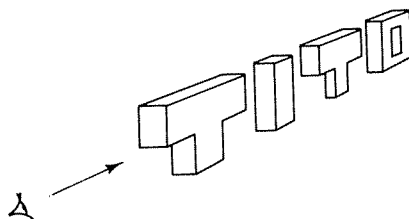


Fig. 11.45

On peut effectuer d'autres rotations si, au lieu de prendre X et Y dans l'ordre, X,Y, on les prend dans l'ordre Y,X, ou si l'ordre de l'autre couple de coordonnées est inversé. Toutes ces rotations, ainsi que beaucoup d'autres, peuvent être réalisées avec la méthode que nous allons présenter, aussi nous ne nous occuperons plus de cette technique.

La première rotation que nous présentons est autour de l'axe des Z. Au paragraphe 8.3 nous avons étudié les équations pour faire tourner le plan X-Y (qui était le seul à l'époque). Si nous appliquons ces équations aux composantes X et Y de chaque point, l'objet dans son entier tournera autour de l'axe des Z. La figure 11.46 montre le sens de la rotation.

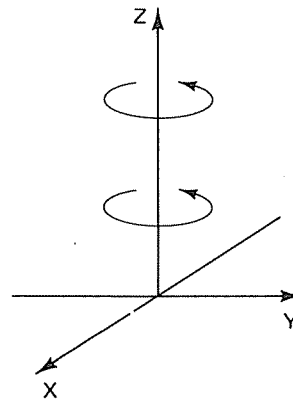


Fig. 11.46

Comme les coordonnées Z de l'objet ne sont pas affectées par cette rotation (la hauteur au-dessus du plan X-Y reste constante), nous pouvons ajouter une troisième équation aux équations 8.1 et 8.2 pour compléter la suite des équations 11.1.

$$\begin{aligned} X1 &= \cos(\text{THETA}) * X + \sin(\text{THETA}) * Y \\ Y1 &= -\sin(\text{THETA}) * X + \cos(\text{THETA}) * Y \\ Z1 &= Z \end{aligned} \quad (11.1)$$

Le programme 11.28 demande l'angle de rotation à la ligne 25 et dessine les lettres pleines de la figure 11.40 avec la rotation appropriée.

```
0 '** 11-28 : Rotation around Z **
10 SCREEN 1:CLS
20 DIM X(60),Y(60),Z(60),LO(100),LD(100)
25 INPUT"Angle ";THETA:
  THETA=THETA*1.745329E-02:CLS
30 FOR I=1 TO 56:
  READ X(I),Y(I),Z(I):
  NEXT
40 FOR I=1 TO 84:
  READ LO(I),LD(I):
  NEXT
50 FOR I=1 TO 84
60  X=X(LO(I)):Y=Y(LO(I)):Z=Z(LO(I))
70  GOSUB 1000
80  XT=X1:YT=Y1
```

```

90  X=X(LD(I)):Y=Y(LD(I)):Z=Z(LD(I))
100  GOSUB 1000
110  LINE(50+XT,140+YT)-(50+X1,140+Y1)
120 NEXT:END
1000 'Rotation subroutine
1010 X1=COS(THETA)*X+SIN(THETA)*Y
1020 Y1=-SIN(THETA)*X+COS(THETA)*Y
1030 Z1=Z:RETURN
5000 DATA 0,-48,0,0,-32,0,16,-32,0,16,0,
          0,32,0,0,32,-32,0,48,-32,0,48,
          -48,0,0,-48,16,0,-32,16
5010 DATA 16,-32,16,16,0,16,32,0,16,32,
          -32,16,48,-32,16,48,-48,16,64,
          -48,0,64,0,0,80,0,0,80,-48,0
5020 DATA 64,-48,16,64,0,16,80,0,16,80,
          -48,16,96,-48,0,96,-32,0,112,
          -32,0,112,0,0,128,0,0,128,-32,0
5030 DATA 144,-32,0,144,-48,0,96,-48,16,
          96,-32,16,112,-32,16,112,0,16,
          128,0,16,128,-32,16,144,-32,16,
          144,-48,16
5040 DATA 160,-48,0,160,0,0,208,0,0,208,-48,
          0,176,-32,0,176,-16,0,192,-16,0,
          192,-32,0,160,-48,16,160,0,16
5050 DATA 208,0,16,208,-48,16,176,-32,16,
          176,-16,16,192,-16,16,192,-32,16
5060 DATA 1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,1
5070 DATA 9,10,10,11,11,12,12,13,13,
          14,14,15,15,16,16,9
5080 DATA 1,9,2,10,3,11,4,12,5,13,6,
          14,7,15,8,16
5090 DATA 17,18,18,19,19,20,20,17
5100 DATA 21,22,22,23,23,24,24,21
5110 DATA 17,21,18,22,19,23,20,24
5120 DATA 25,26,26,27,27,28,28,29,
          29,30,30,31,31,32,32,25
5130 DATA 33,34,34,35,35,36,36,37,
          37,38,38,39,39,40,40,33
5140 DATA 25,33,26,34,27,35,28,36,
          29,37,30,38,31,39,32,40
5150 DATA 41,42,42,43,43,44,44,41,
          49,50,50,51,51,52,52,49
5160 DATA 45,46,46,47,47,48,48,45,
          53,54,54,55,55,56,56,53
5170 DATA 41,49,42,50,43,51,44,52,
          45,53,46,54,47,55,48,56

```

PROGRAMME 11.28

La figure 11.47 montre le résultat avec un angle de 25 degrés.

Nous allons étudier maintenant comment réaliser une rotation autour de l'axe des X. La figure 11.48 montre le sens de cette rotation.

L'ensemble des équations pour faire tourner un point autour de l'axe des X d'un angle THETA est :

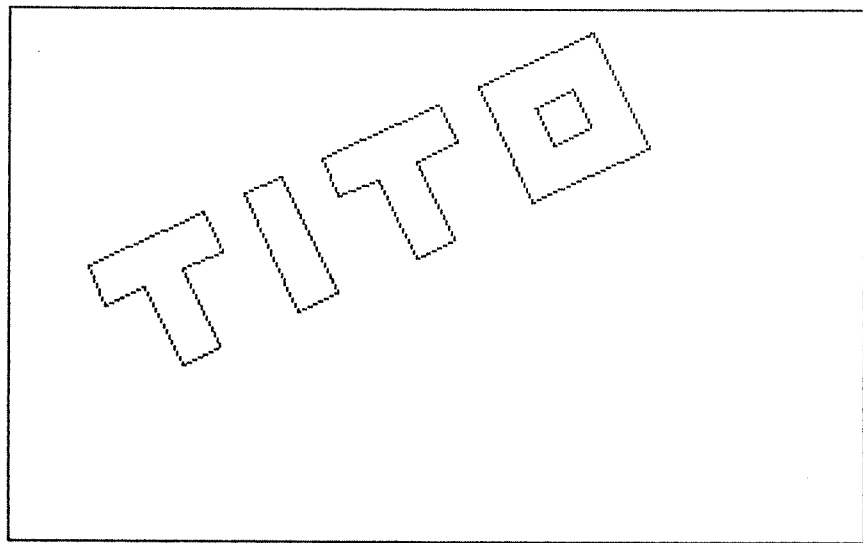


Fig. 11.47

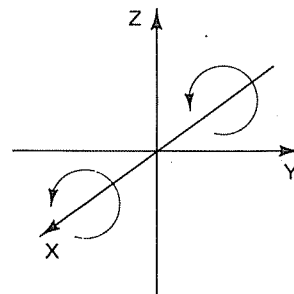


Fig. 11.48

$$X1 = X$$

$$Y1 = Y \cdot \cos(\text{THETA}) - Z \cdot \sin(\text{THETA}) \quad (11.2)$$

$$Z1 = Y \cdot \sin(\text{THETA}) + Z \cdot \cos(\text{THETA})$$

Le programme 11.29 fait tourner les lettres pleines du programme 11.27 autour de l'axe des X, l'angle étant entré en ligne 25. La figure 11.49 montre les lettres ayant tourné de 80 degrés. Notez que c'est la première fois que la composante Z a été prise en considération.

```
0  *** 11-29 : Rotation around X **
10 SCREEN 1:CLS
20 DIM X(60),Y(60),Z(60),LD(100),LD(100)
25 INPUT"Angle ";THETA:
```

```

    THETA=THETA*1.745329E-02:CLS
30 FOR I=1 TO 56:
    READ X(I),Y(I),Z(I):
    NEXT
40 FOR I=1 TO 84:
    READ LO(I),LD(I):
    NEXT
50 FOR I=1 TO 84
60   X=X(LO(I)):Y=Y(LO(I)):Z=Z(LO(I))
70   GOSUB 1000
80   XT=X1:YT=Y1
90   X=X(LD(I)):Y=Y(LD(I)):Z=Z(LD(I))
100  GOSUB 1000
110  LINE(50+XT,140+YT)-(50+X1,140+Y1)
120 NEXT:END
1000 'Rotation subroutine
1010 X1=X
1020 Y1=Y*COS(THETA)-Z*SIN(THETA)
1030 Z1=Y*SIN(THETA)+Z*COS(THETA)
1040 RETURN
5000 DATA 0,-48,0,0,-32,0,16,-32,0,16,0,
          0,32,0,0,32,-32,0,48,-32,0,48,
          -48,0,0,-48,16,0,-32,16
5010 DATA 16,-32,16,16,0,16,32,0,16,32,
          -32,16,48,-32,16,48,-48,16,64,
          -48,0,64,0,0,80,0,0,80,-48,0
5020 DATA 64,-48,16,64,0,16,80,0,16,80,
          -48,16,96,-48,0,96,-32,0,112,
          -32,0,112,0,0,128,0,0,128,-32,0
5030 DATA 144,-32,0,144,-48,0,96,-48,16,
          96,-32,16,112,-32,16,112,0,16,
          128,0,16,128,-32,16,144,-32,16,
          144,-48,16
5040 DATA 160,-48,0,160,0,0,208,0,0,208,-48,
          0,176,-32,0,176,-16,0,192,-16,0,
          192,-32,0,160,-48,16,160,0,16
5050 DATA 208,0,16,208,-48,16,176,-32,16,
          176,-16,16,192,-16,16,192,-32,16
5060 DATA 1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,1
5070 DATA 9,10,10,11,11,12,12,13,13,
          14,14,15,15,16,16,9
5080 DATA 1,9,2,10,3,11,4,12,5,13,6,
          14,7,15,8,16
5090 DATA 17,18,18,19,19,20,20,17
5100 DATA 21,22,22,23,23,24,24,21
5110 DATA 17,21,18,22,19,23,20,24
5120 DATA 25,26,26,27,27,28,28,29,
          29,30,30,31,31,32,32,25
5130 DATA 33,34,34,35,35,36,36,37,
          37,38,38,39,39,40,40,33
5140 DATA 25,33,26,34,27,35,28,36,
          29,37,30,38,31,39,32,40
5150 DATA 41,42,42,43,43,44,44,41,
          49,50,50,51,51,52,52,49
5160 DATA 45,46,46,47,47,48,48,45,
          53,54,54,55,55,56,56,53
5170 DATA 41,49,42,50,43,51,44,52,
          45,53,46,54,47,55,48,56

```

PROGRAMME 11.29

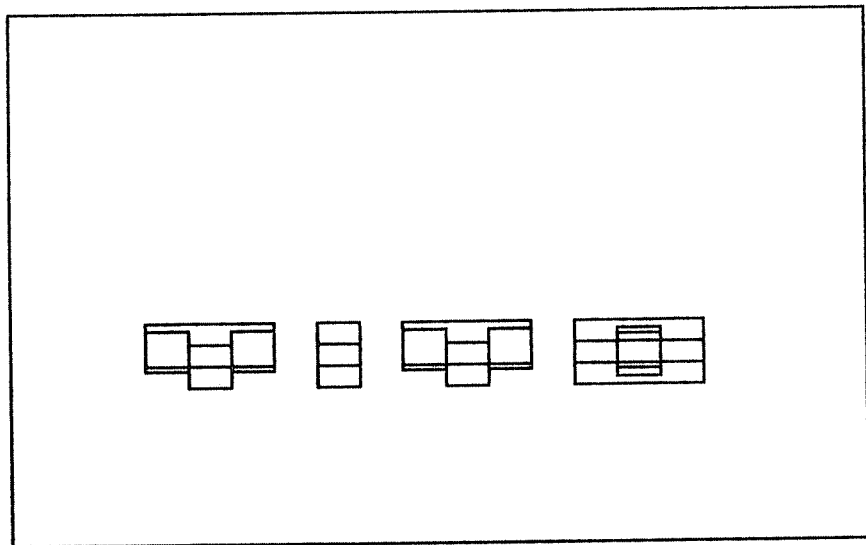


Fig. 11.49

Finalement, la figure 11.50 montre le sens de la rotation autour de l'axe des Y.

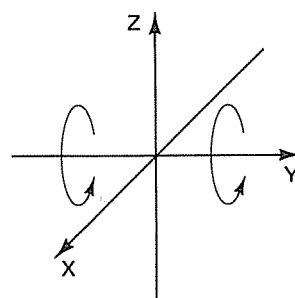


Fig. 11.50

Les équations pour faire tourner un point autour de l'axe des Y d'un angle THETA sont :

$$\begin{aligned} X1 &= X \cdot \cos(\text{THETA}) + Z \cdot \sin(\text{THETA}) \\ Y1 &= Y \\ Z1 &= -X \cdot \sin(\text{THETA}) + Z \cdot \cos(\text{THETA}) \end{aligned} \quad (11.3)$$

Le programme 11.30 utilise ces équations pour faire tourner les lettres du programme 11.27 autour de l'axe des Y.


```

0  '** 11-30 : Rotation around Y **
10 SCREEN 1:CLS
20 DIM X(60),Y(60),Z(60),LD(100),LD(100)
25 INPUT"Angle ";THETA:
   THETA=THETA*1.745329E-02:CLS
30 FOR I=1 TO 56:
   READ X(I),Y(I),Z(I):
   NEXT
40 FOR I=1 TO 84:
   READ LD(I),LD(I):
   NEXT
50 FOR I=1 TO 84
60   X=X(LD(I)):Y=Y(LD(I)):Z=Z(LD(I))
70   GOSUB 1000
80   XT=X1:YT=Y1
90   X=X(LD(I)):Y=Y(LD(I)):Z=Z(LD(I))
100  GOSUB 1000
110  LINE(50+XT,140+YT)-(50+X1,140+Y1)
120 NEXT:END
1000 'Rotation subroutine
1010 X1=X*COS(THETA)+Z*SIN(THETA)
1020 Y1=Y
1030 Z1=-X*SIN(THETA)+Z*COS(THETA)
1040 RETURN
5000 DATA 0,-48,0,0,-32,0,16,-32,0,16,0,
        0,32,0,0,32,-32,0,48,-32,0,48,
        -48,0,0,-48,16,0,-32,16
5010 DATA 16,-32,16,16,0,16,32,0,16,32,
        -32,16,48,-32,16,48,-48,16,64,
        -48,0,64,0,0,80,0,0,80,-48,0
5020 DATA 64,-48,16,64,0,16,80,0,16,80,
        -48,16,96,-48,0,96,-32,0,112,
        -32,0,112,0,0,128,0,0,128,-32,0
5030 DATA 144,-32,0,144,-48,0,96,-48,16,
        96,-32,16,112,-32,16,112,0,16,
        128,0,16,128,-32,16,144,-32,16,
        144,-48,16
5040 DATA 160,-48,0,160,0,0,208,0,0,208,-
        0,176,-32,0,176,-16,0,192,-16,0
        192,-32,0,160,-48,16,160,0,16
5050 DATA 208,0,16,208,-48,16,176,-32,16,
        176,-16,16,192,-16,16,192,-32,1
5060 DATA 1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,1
5070 DATA 9,10,10,11,11,12,12,13,13,
        14,14,15,15,16,16,9
5080 DATA 1,9,2,10,3,11,4,12,5,13,6,
        14,7,15,8,16
5090 DATA 17,18,18,19,19,20,20,17
5100 DATA 21,22,22,23,23,24,24,21
5110 DATA 17,21,18,22,19,23,20,24
5120 DATA 25,26,26,27,27,28,28,29,
        29,30,30,31,31,32,32,25
5130 DATA 33,34,34,35,35,36,36,37,
        37,38,38,39,39,40,40,33
5140 DATA 25,33,26,34,27,35,28,36,
        29,37,30,38,31,39,32,40
5150 DATA 41,42,42,43,43,44,44,41,
        49,50,50,51,51,52,52,49
5160 DATA 45,46,46,47,47,48,48,45,
        53,54,54,55,55,56,56,53

```

```
5170 DATA 41,49,42,50,43,51,44,52,
          45,53,46,54,47,55,48,56
```

PROGRAMME 11.30

La figure 11.51 montre les lettres ayant tourné de 25 degrés. Comme avec toutes les vues de face (qui sont les seules que nous avons réalisées jusqu'ici), l'effet de relief n'est pas extraordinaire, mais nous allons résoudre ce problème avec le programme suivant.

Nous allons combiner les trois rotations dans un simple sous-programme. Les angles utilisés seront THETA, RHO et PHI. Le programme 11.31 demande à la ligne 25 la valeur des trois angles et fait tourner les lettres du programme 11.27 en conséquence. Notez que les coordonnées de chaque point sont d'abord traitées avec les équations 11.1 pour avoir les valeurs X1, Y1 et Z1, celles-ci sont traitées avec les équations 11.2 pour avoir X2, Y2 et Z2 et finalement avec les équations 11.3 pour avoir X3, Y3 et Z3.

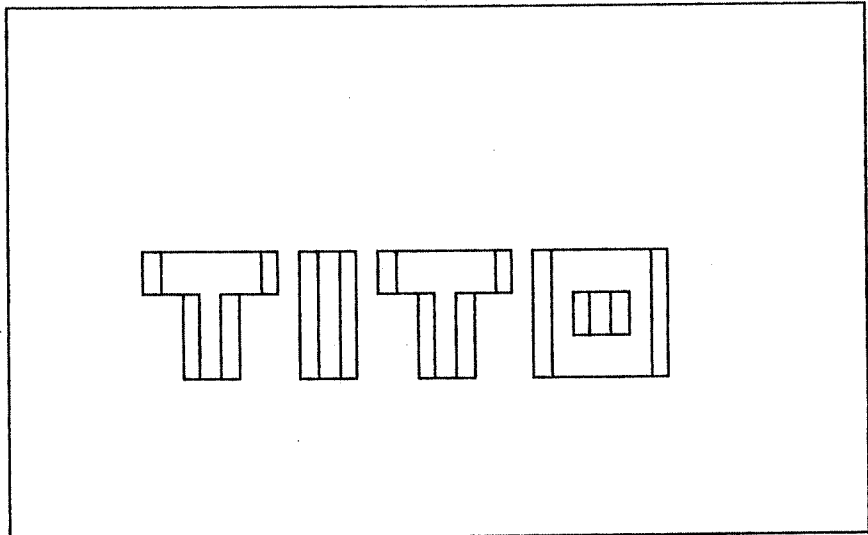


Fig. 11.51

```
0 *** 11-31 : Rotation around the 3 axes **
5 F=1.745329E-02
10 SCREEN 1:CLS
20 DIM X(60),Y(60),Z(60),LD(100),LD(100)
25 INPUT "Angles (Axes: Z, X, Y) ";THETA,RHO,PHI
27 THETA=THETA*F:RHO=RHO*F:PHI=PHI*F
29 CLS
30 FOR I=1 TO 56:
    READ X(I),Y(I),Z(I):
    NEXT
40 FOR I=1 TO 84:
```

```

      READ LO(I),LD(I):
    NEXT
50 FOR I=1 TO 84
60   X=X(LO(I)):Y=Y(LO(I)):
      Z=Z(LO(I)):GOSUB 1000
70   XT=X3:YT=Y3
80   X=X(LD(I)):Y=Y(LD(I)):
      Z=Z(LD(I)):GOSUB 1000
90   LINE(50+XT,100+YT)-(50+X3,100+Y3)
100 NEXT
110 END
1000 ** Rotation subroutine **
1010 X1=X*COS(PHI)+Z*SIN(PHI)
1020 Y1=Y
1030 Z1=-X*SIN(PHI)+Z*COS(PHI)
1040 X2=X1*COS(THETA)+Y1*SIN(THETA)
1050 Y2=-X1*SIN(THETA)+Y1*COS(THETA)
1060 Z2=Z1
1070 X3=X2
1080 Y3=Y2*COS(RHO)-Z2*SIN(RHO)
1090 Z3=Y2*SIN(RHO)+Z2*COS(RHO)
1100 RETURN
5000 DATA 0,-48,0,0,-32,0,16,-32,0,16,0,
           0,32,0,0,32,-32,0,48,-32,0,48,
           -48,0,0,-48,16,0,-32,16
5010 DATA 16,-32,16,16,0,16,32,0,16,32,
           -32,16,48,-32,16,48,-48,16,64,
           -48,0,64,0,0,80,0,0,80,-48,0
5020 DATA 64,-48,16,64,0,16,80,0,16,80,
           -48,16,96,-48,0,96,-32,0,112,
           -32,0,112,0,0,128,0,0,128,-32,0
5030 DATA 144,-32,0,144,-48,0,96,-48,16,
           96,-32,16,112,-32,16,112,0,16,
           128,0,16,128,-32,16,144,-32,16,
           144,-48,16
5040 DATA 160,-48,0,160,0,0,208,0,0,208,-48,
           0,176,-32,0,176,-16,0,192,-16,0,
           192,-32,0,160,-48,16,160,0,16
5050 DATA 208,0,16,208,-48,16,176,-32,16,
           176,-16,16,192,-16,16,192,-32,16
5060 DATA 1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,1
5070 DATA 9,10,10,11,11,12,12,13,13,
           14,14,15,15,16,16,9
5080 DATA 1,9,2,10,3,11,4,12,5,13,6,
           14,7,15,8,16
5090 DATA 17,18,18,19,19,20,20,17
5100 DATA 21,22,22,23,23,24,24,21
5110 DATA 17,21,18,22,19,23,20,24
5120 DATA 25,26,26,27,27,28,28,29,
           29,30,30,31,31,32,32,25
5130 DATA 33,34,34,35,35,36,36,37,
           37,38,38,39,39,40,40,33
5140 DATA 25,33,26,34,27,35,28,36,
           29,37,30,38,31,39,32,40
5150 DATA 41,42,42,43,43,44,44,41,
           49,50,50,51,51,52,52,49
5160 DATA 45,46,46,47,47,48,48,45,
           53,54,54,55,55,56,56,53
5170 DATA 41,49,42,50,43,51,44,52,
           45,53,46,54,47,55,48,56

```

PROGRAMME 11.31

La figure 11.52 montre les lettres tournées avec les angles THETA = 0, RHO = 30 et PHI = 30.

L'ordre dans lequel les rotations sont accomplies influe sur le résultat. Si le sous-programme 1000 du programme est changé en celui que l'on voit dans le programme 11.32, les angles utilisés à la figure 11.52 produisent le graphique de la figure 11.53.

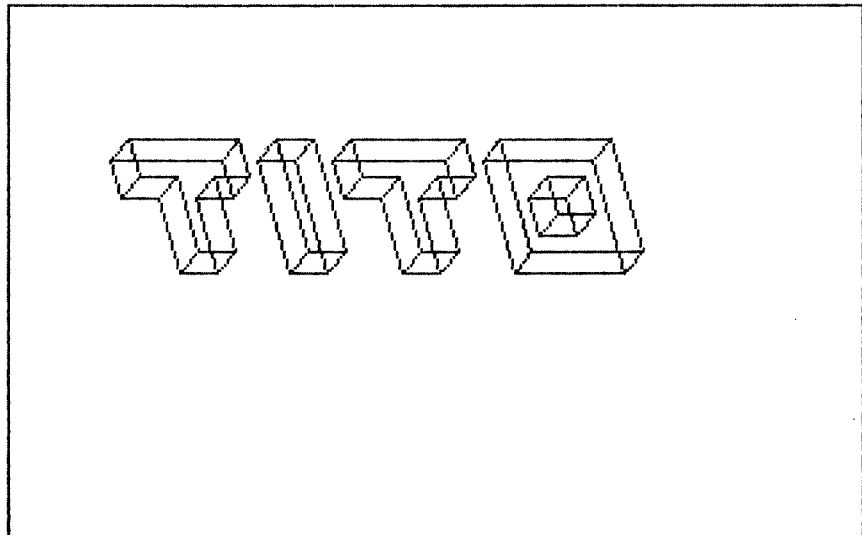


Fig. 11.52

```

1000 '** Rotation subroutine **
1010 X1=X
1020 Y1=Y*COS(RHO)-Z*SIN(RHO)
1030 Z1=Y*SIN(RHO)+Z*COS(RHO)
1040 X2=X1*COS(THETA)+Y1*SIN(THETA)
1050 Y2=-X1*SIN(THETA)+Y1*COS(THETA)
1060 Z2=Z1
1070 X3=X2*COS(PHI)+Z2*SIN(PHI)
1080 Y3=Y2
1090 Z3=-X2*SIN(PHI)+Z2*COS(PHI)
1100 RETURN

```

PROGRAMME 11.32

Nous allons maintenant étudier un programme qui permet la rotation interactive d'un objet défini à l'aide d'un algorithme. En utilisant les coordonnées polaires, le programme 11.33 trace un octogone de rayon 80. A la ligne 30, Z est fixé à 0, ainsi l'octogone est dans le plan X-Y. Chaque X et chaque Y calculés sont envoyés au sous-programme 1000 (le sous-programme de rotation) et les lignes correspondantes sont tracées. Dans la boucle des lignes 140 à 210, les points de l'octogone sont reliés au point (0,0,40), qui est le point situé 40 unités au-dessus du

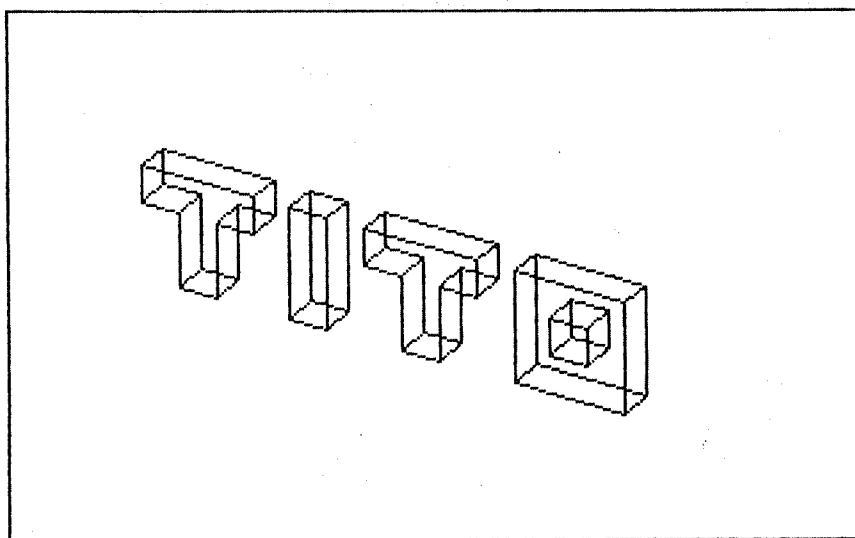


Fig. 11.53

centre du plan X-Y. La figure résultante ressemble à une antenne radar. Si on appuie sur une des touches de contrôle du curseur, ce sera détecté soit par le INKEY\$ de la ligne 70, soit par le INPUT\$(1) de la ligne 500. Les touches ont l'effet décrit à la table 11.3. Les figures 11.54 à 11.57 montrent les graphiques réalisés avec différents angles de rotation.

Table 11.3

TOUCHE	EFFET
8	Décrémente l'angle RHO de dix degrés
2	Incrémente l'angle RHO de dix degrés
4	Décrémente l'angle PHI de dix degrés
6	Incrémente l'angle PHI de dix degrés
7	Décrémente l'angle THETA de dix degrés
9	Incrémente l'angle THETA de dix degrés

```

0 '** 11-33 : Interactive rotation **
5 F=1.745329E-02
10 SCREEN 1:CLS
20 R=80
30 Z=0
32 FOR A=0 TO 360 STEP 45
35   AN=A*F
40   X=R*COS(AN)
50   Y=R*SIN(AN)
60   GOSUB 1000
70   W$=INKEY$:
   IF W$<>" "

```

```

        THEN
            510
120 IF A=0
    THEN
        PSET(160+X3,100+Y3)
    ELSE
        LINE-(160+X3,100+Y3)
130 NEXT
140 FOR A=0 TO 360 STEP 45
145   AN=A*F
150   X=R*COS(AN)
160   Y=R*SIN(AN)
170   Z=0:GOSUB 1000
175   W$=INKEY$:
        IF W$<>" "
            THEN
                510
180   XT=X3:YT=Y3
190   X=0:Y=0:Z=40:GOSUB 1000
200   LINE(160+XT,100+YT)-(160+X3,100+Y3),2
210 NEXT
500 W$=INPUT$(1)
510 V=VAL(W$)
520 IF V=8
    THEN
        RHO=RHO-10:CLS:GOTO 30
530 IF V=2
    THEN
        RHO=RHO+10:CLS:GOTO 30
540 IF V=4
    THEN
        PHI=PHI-10:CLS:GOTO 30
550 IF V=6
    THEN
        PHI=PHI+10:CLS:GOTO 30
554 IF V=7
    THEN
        THETA=THETA-10:CLS:GOTO 30
557 IF V=9
    THEN
        THETA=THETA+10:CLS:GOTO 30
560 BEEP:GOTO 32
1000 '** Rotations subroutine **
1001 SR=SIN(RHO*F):CR=COS(RHO*F)
1002 ST=SIN(THETA*F):CT=COS(THETA*F)
1003 SP=SIN(PHI*F):CP=COS(PHI*F)
1010 X1=X
1020 Y1=Y*CR-Z*SR
1030 Z1=Y*SR+Z*CR
1040 X2=X1*CT+Y1*ST
1050 Y2=-X1*ST+Y1*CT
1060 Z2=Z1
1070 X3=X2*CP+Z2*SP
1080 Y3=Y2
1090 Z3=-X2*SP+Z2*CP
1100 RETURN

```

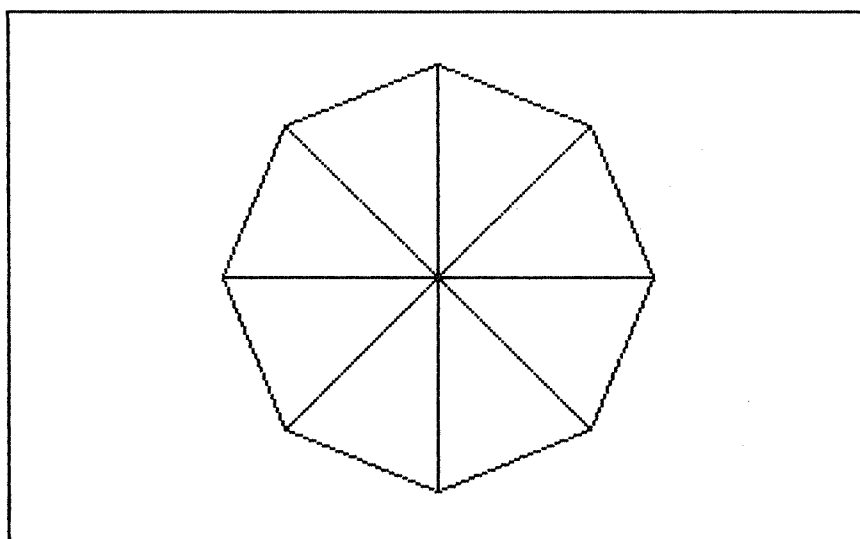


Fig. 11.54

Le programme 11.34 fait tourner la figure rentrée sous forme vectorielle aux lignes 40 à 130. (Tapez 999 pour X ou l'origine, et des zéros à la suite pour terminer l'entrée). Les touches pour changer les angles de rotation sont celles de la table 11.3. Essayez avec le cube décrit par les tables 11.2 et 11.1, et la maison dont les données figurent à la table 11.4.

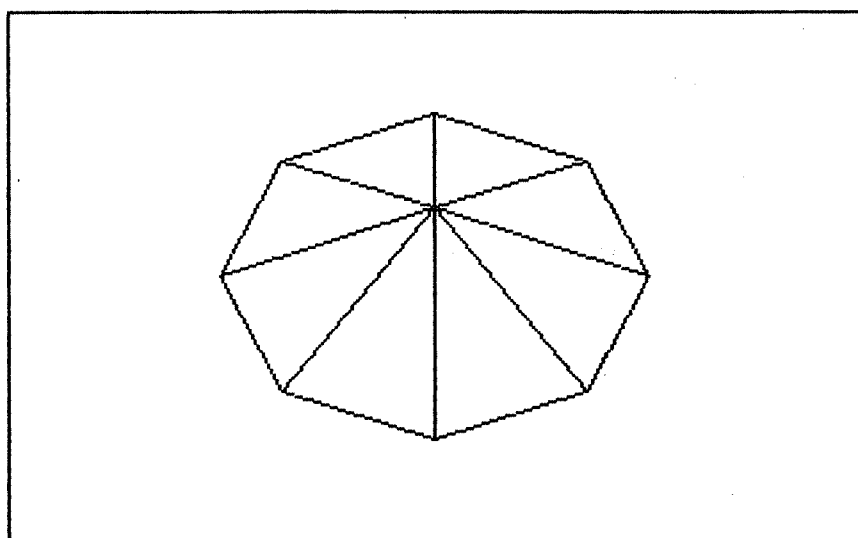


Fig. 11.55

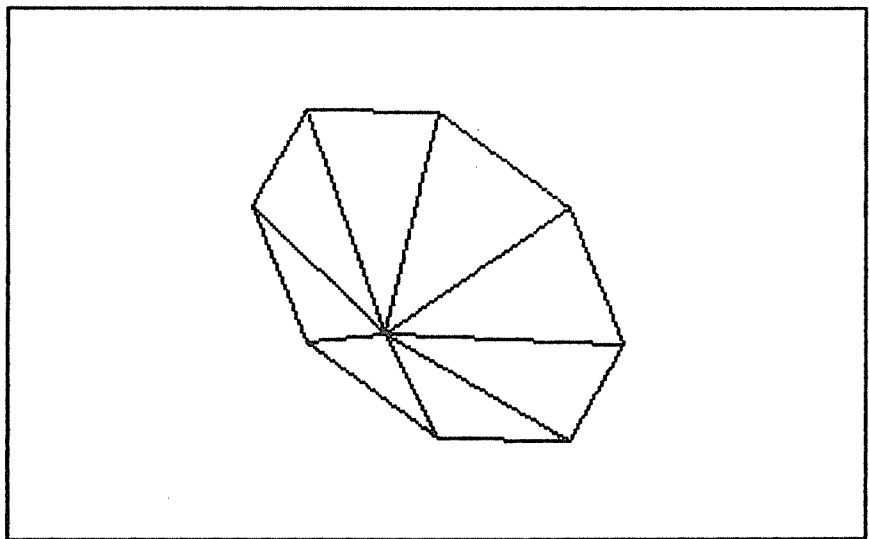


Fig. 11.56

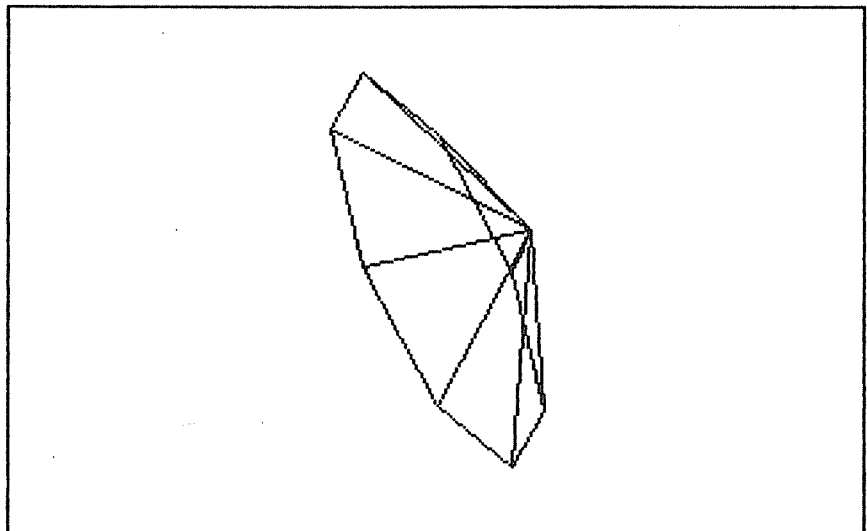


Fig. 11.57

```

0 '** 11-34 : Interactive rotation **
5 '      ** of object in vector form **
10 F=1.745329E-02
20 SCREEN 1:CLS
30 DIM X(100),Y(100),Z(100),LD(100),LD(100)
40 FOR I=1 TO 100
50   PRINT"Point #";I;"(X,Y,Z) ";

```



```

60 INPUT X(I),Y(I),Z(I)
70 IF X(I)=999 THEN NP=I-1:GOTO 90
80 NEXT:NP=I-1
90 FOR I=1 TO 100
100 PRINT"Line #";I;"(Origin,destination)";
110 INPUT LO(I),LD(I)
120 IF LO(I)=999
    THEN
        NL=I-1:GOTO 140
130 NEXT:NL=I-1
140 CLS
150 FOR I=1 TO NL
160 W$=INKEY$:
    IF W$<>" "
        THEN
            510
170 X=X(LO(I)):Y=Y(LO(I)):Z=Z(LO(I))
180 GOSUB 1000
190 XT=X3:YT=Y3
200 X=X(LD(I)):Y=Y(LD(I)):Z=Z(LD(I))
210 GOSUB 1000
220 LINE(160+XT,100+YT)-(160+X3,100+Y3)
230 NEXT
500 W$=INPUT$(1)
510 V=VAL(W$)
520 IF V=8
    THEN
        RHO=RHO-10:CLS:GOTO 140
530 IF V=2
    THEN
        RHO=RHO+10:CLS:GOTO 140
540 IF V=4
    THEN
        PHI=PHI-10:CLS:GOTO 140
550 IF V=6
    THEN
        PHI=PHI+10:CLS:GOTO 140
554 IF V=7
    THEN
        THETA=THETA-10:CLS:GOTO 140
557 IF V=9
    THEN
        THETA=THETA+10:CLS:GOTO 140
560 BEEP:GOTO 140
1000 '** Rotation subroutine **
1010 SR=SIN(RHO*F):CR=COS(RHO*F)
1020 ST=SIN(THETA*F):CT=COS(THETA*F)
1030 SP=SIN(PHI*F):CP=COS(PHI*F)
1040 X1=X
1050 Y1=Y*CR-Z*SR
1060 Z1=Y*SR+Z*CR
1070 X2=X1*CT+Y1*ST
1080 Y2=-X1*ST+Y1*CT
1090 Z2=Z1
1100 X3=X2*CP+Z2*SP
1110 Y3=Y2
1120 Z3=-X2*SP+Z2*CP
1130 RETURN

```

PROGRAMME 11.34

Table 11.4

POINT	X	Y	Z
1	0	0	0
2	60	0	0
3	60	60	0
4	30	90	0
5	0	60	0
6	0	0	60
7	60	0	60
8	60	60	60
9	30	90	60
10	0	60	60

LIGNE	ORIGINE	DESTINATION
1	1	2
2	2	3
3	3	4
4	4	5
5	5	1
6	3	5
7	6	7
8	7	8
9	8	9
10	9	10
11	10	6
12	8	10
13	1	6
14	2	7
15	3	8
16	4	9
17	5	10

Les figures 11.58 à 11.61 montrent la maison selon quatre vues différentes après rotations.

11.2.4 La perspective

Le type de projection que nous avons utilisé depuis le paragraphe 11.2.1 jusqu'à maintenant ignore complètement la perspective et suppose un point de fuite situé à l'infini. Ce type de projection est appelé projection orthographique ou isométrique, parce qu'elle conserve la taille des objets sans se soucier de leur distance par rapport à l'observateur. Dans la photographie, ce type de perspective peut être approché par

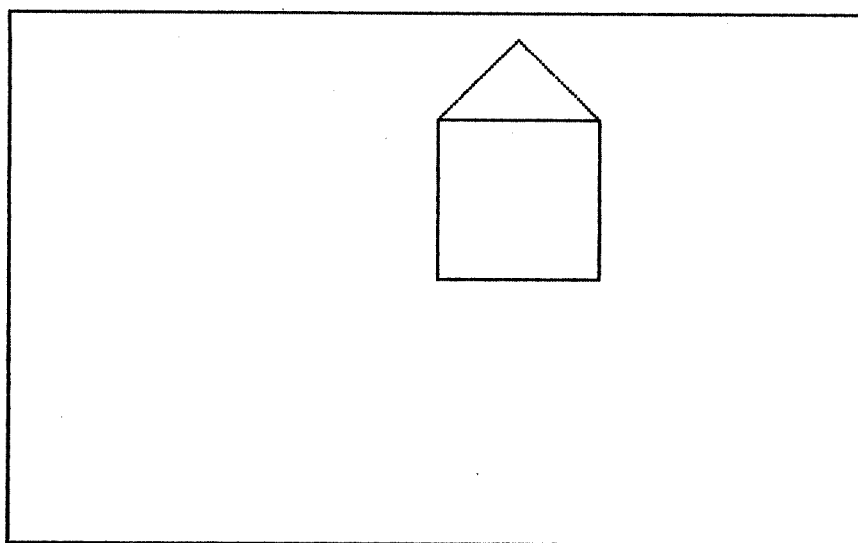


Fig. 11.58

un téléobjectif puissant et une grande distance entre la caméra et l'objet.

En réalité, la taille apparente des objets est proportionnelle à leur distance à l'observateur, comme nous l'avons expliqué au paragraphe 11.1.10, où nous avons expliqué les concepts de point de fuite et de

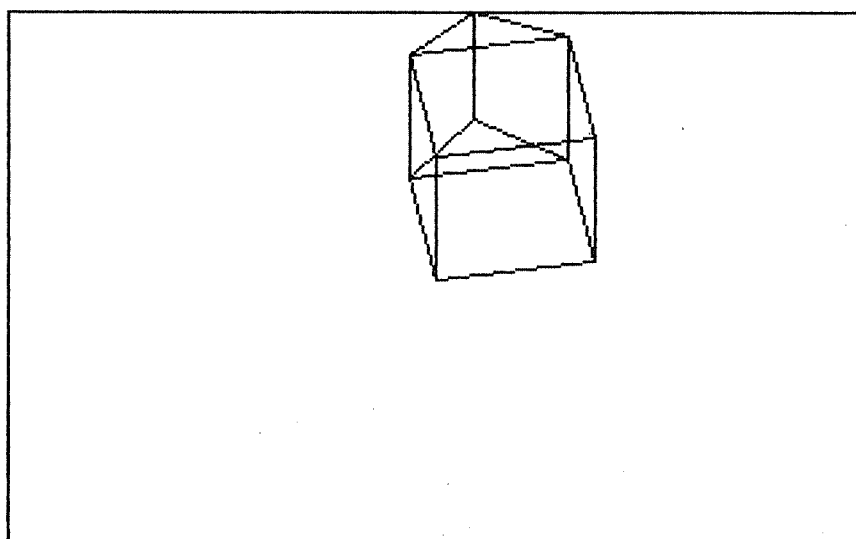


Fig. 11.59

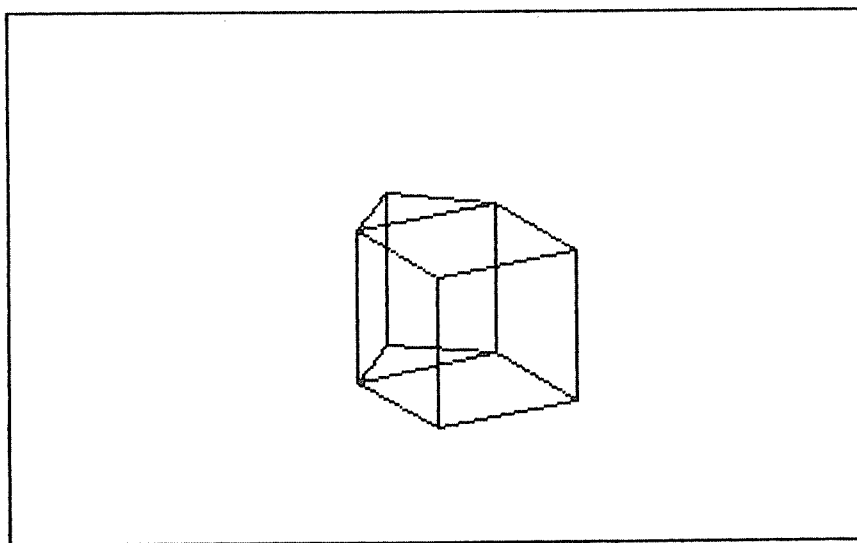


Fig. 11.60

lignes convergentes. Bien qu'on ait l'habitude de dire qu'un point de fuite se trouve à l'infini, nous verrons que dans notre représentation c'est un point fixé arbitrairement pour déterminer la taille relative des objets, et les facteurs de perspective. La figure 11.62 montre un cube vu avec et sans perspective.

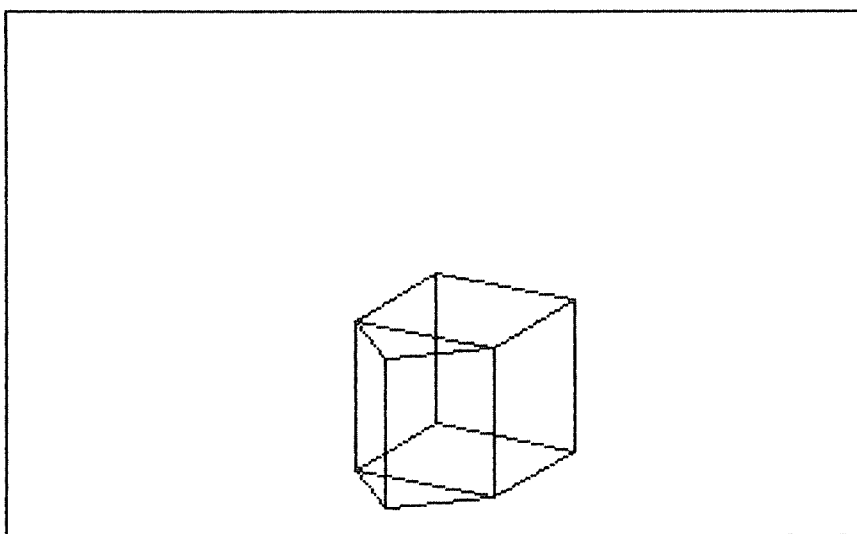


Fig. 11.61

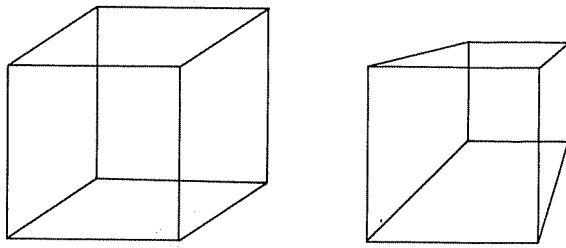


Fig. 11.62

La profondeur. Dans ce paragraphe, nous étudierons le type de perspective le plus apparent, la perspective causée par la profondeur. La figure 11.63 montre la relation d'un cube (vu de dessus) avec un observateur situé à une distance d . L'angle formé par les points A,B et l'observateur est plus grand que celui formé par les points C,D et l'observateur. Ceci fait que la distance C-D a l'air plus petite que la distance A-B, comme on peut voir à la figure 11.63. Les lignes qui relient le carré proche et le carré distant convergent lentement, provoquant une sensation naturelle de perspective. Ceci est le type d'effet produit par l'objectif normal ou standard d'un appareil photo.

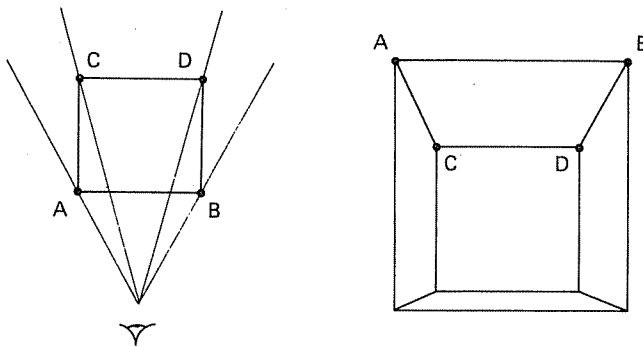


Fig. 11.63

La figure 11.64 montre un cube vu par un observateur situé à une distance d' , plus petite que d . La proportion des angles compris entre les points A,B et l'observateur, et les points C,D et l'observateur a changé, rendant la distance C-D beaucoup plus petite que A-B, comme on peut voir sur la figure 11.64. Notez que les lignes reliant les carrés proche et distant convergent très rapidement, produisant un effet de perspective plus prononcé, du type produit par les objectifs grand angle des appareils photo.

A partir de l'explication précédente, nous pouvons voir que la perspective est en fait proportionnelle à la distance et à la dimension des ob-

jets observés. Comme nous l'avons vu au paragraphe 11.1.10, il est possible de choisir la distance relative entre le point le plus éloigné de l'objet et le point de fuite, notre infini hypothétique.

Dans une vue en perspective, les objets qui sont plus proches de l'observateur ont l'air plus grands et les objets distants ont l'air plus petits. Nous choisirons de représenter les objets les plus proches de l'observateur à leur dimension normale (sans perspective), les objets situés au point de fuite à la dimension zéro, et les objets situés entre ces deux extrêmes à une dimension proportionnelle à leur éloignement. Nous pouvons utiliser la fonction du paragraphe 11.1.10 pour calculer le facteur d'échelle pour chaque point, tel que :

FACTOR = 0 Si le point est au point de fuite

FACTOR = 1 Si le point est le plus proche de l'observateur

Nous redonnons ci-dessous les formules calculées au paragraphe 11.1.10,

où

$$\text{FACTOR} = (Z - \text{VANISHING.POINT}) / \text{DISTANCE},$$

$$\text{DISTANCE} = \text{MAXIMUM.Z} - \text{VANISHING.POINT}.$$

Notez qu'ici le facteur dépend de la distance du point au plan X-Y, c'est-à-dire de la coordonnée Z. Une fois le facteur d'échelle calculé, les coordonnées X et Y de chaque point doivent être multipliées par lui pour trouver les coordonnées en perspective.

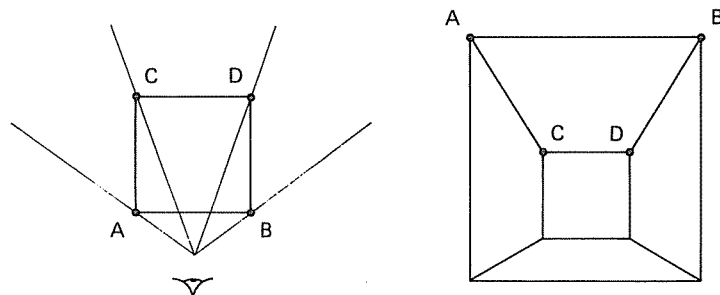


Fig. 11.64

Exemple.

Les points d'un objet ont les coordonnées Z suivantes :

POINT	Z
P1	0
P2	4
P3	77
P4	105

Le point de fuite est en $Z = -100$, donc les facteurs d'échelle sont calculés ainsi : $DISTANCE = 105 - (-100) = 205$. Pour chaque point les coordonnées X et Y doivent être multipliées par :

Au point P1, $FACTOR = (0 - (-100)) / 205 = .4878049$

Au point P2, $FACTOR = (4 - (-100)) / 205 = .5073171$

Au point P3, $FACTOR = (77 - (-100)) / 205 = .8634146$

Au point P4, $FACTOR = (105 - (-100)) / 205 = 1$

Le programme 11.35 demande les angles de rotation à la ligne 30, et le point de fuite à la ligne 40. Il trace les lettres de la figure 11.40, cette fois en perspective.

```

0  '** 11-35 : Depth perspective **
10 F=1.745329E-02
15 NP=56:NL=84
20 SCREEN 1:CLS
30 INPUT"Rotation angles (around X,Y) ";B,C
40 INPUT"Vanishing point ";VANISHING.POINT:CLS
50 DIM X(60),Y(60),Z(60),LD(100),LD(100)
60 FOR I=1 TO NP:
    READ X(I),Y(I),Z(I):
    X(I)=X(I)-90:Y(I)=Y(I)+23:
    NEXT
70 FOR I=1 TO NL:
    READ LD(I),LD(I):
    NEXT
80 MAXIMUM.Z=-10000
90 FOR I=1 TO NP
110  IF Z(I)>MAXIMUM.Z
    THEN
        MAXIMUM.Z=Z(I)
120 NEXT
130 DISTANCE=MAXIMUM.Z-VANISHING.POINT
140 FOR I=1 TO NL
150  X=X(LD(I)):Y=Y(LD(I)):Z=Z(LD(I))
155  GOSUB 2000:GOSUB 1000
160  XT=X2:YT=Y2
170  X=X(LD(I)):Y=Y(LD(I)):Z=Z(LD(I))
175  GOSUB 2000:GOSUB 1000
180  LINE(140+XT,100+YT)-(140+X2,100+Y2)
190 NEXT
200 END

```

```

1000 '** Rotation subroutine **
1010 CB=COS(B*F):SB=SIN(B*F)
1020 CC=COS(C*F):SC=SIN(C*F)
1030 X1=X*CB+Z*SB
1040 Y1=Y
1050 Z1=-X*SB+Z*CB
1060 X2=X1
1070 Y2=Y1*CC-Z1*SC
1080 Z2=Y1*SC+Z1*CC
1090 RETURN
2000 '** Perspective subroutine **
2010 FACTOR=(Z-VANISHING.POINT)/DISTANCE
2020 X=X*FACTOR:Y=Y*FACTOR
2030 RETURN
5000 DATA 0,-48,0,0,-32,0,16,-32,0,16,0,
          0,32,0,0,32,-32,0,48,-32,0,48,
          -48,0,0,-48,16,0,-32,16
5010 DATA 16,-32,16,16,0,16,32,0,16,32,
          -32,16,48,-32,16,48,-48,16,64,
          -48,0,64,0,0,80,0,0,80,-48,0
5020 DATA 64,-48,16,64,0,16,80,0,16,80,
          -48,16,96,-48,0,96,-32,0,112,
          -32,0,112,0,0,128,0,0,128,-32,0
5030 DATA 144,-32,0,144,-48,0,96,-48,16,
          96,-32,16,112,-32,16,112,0,16,
          128,0,16,128,-32,16,144,-32,16,
          144,-48,16
5040 DATA 160,-48,0,160,0,0,208,0,0,208,-48,
          0,176,-32,0,176,-16,0,192,-16,0,
          192,-32,0,160,-48,16,160,0,16
5050 DATA 208,0,16,208,-48,16,176,-32,16,
          176,-16,16,192,-16,16,192,-32,16
5060 DATA 1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,1
5070 DATA 9,10,10,11,11,12,12,13,13,
          14,14,15,15,16,16,9
5080 DATA 1,9,2,10,3,11,4,12,5,13,6,
          14,7,15,8,16
5090 DATA 17,18,18,19,19,20,20,17
5100 DATA 21,22,22,23,23,24,24,21
5110 DATA 17,21,18,22,19,23,20,24
5120 DATA 25,26,26,27,27,28,28,29,
          29,30,30,31,31,32,32,25
5130 DATA 33,34,34,35,35,36,36,37,
          37,38,38,39,39,40,40,33
5140 DATA 25,33,26,34,27,35,28,36,
          29,37,30,38,31,39,32,40
5150 DATA 41,42,42,43,43,44,44,41,
          49,50,50,51,51,52,52,49
5160 DATA 45,46,46,47,47,48,48,45,
          53,54,54,55,55,56,56,53
5170 DATA 41,49,42,50,43,51,44,52,
          45,53,46,54,47,55,48,56

```

PROGRAMME 11.35

La figure 11.65 montre le graphique produit avec des angles d'inclinaison 0,0, et un point de fuite à -150.

La figure 11.66 montre les mêmes lettres avec des angles 0,0, et un point de fuite à -30.

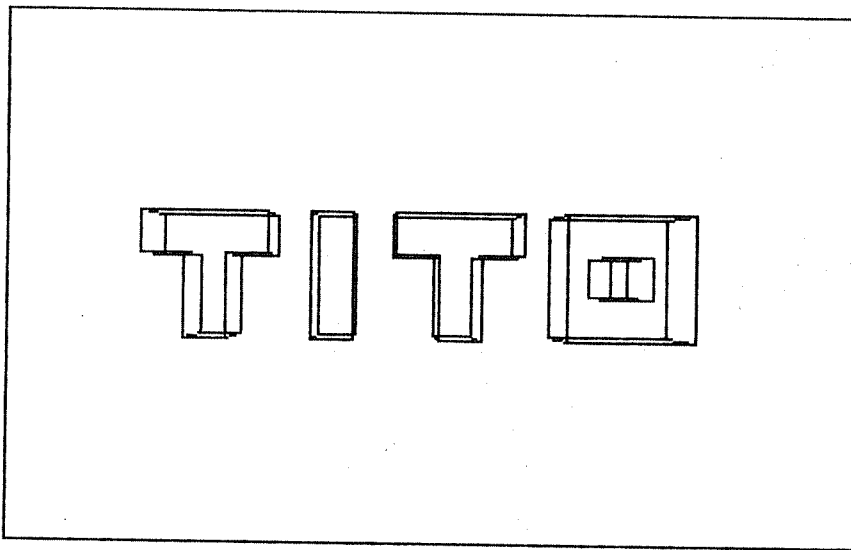


Fig. 11.65

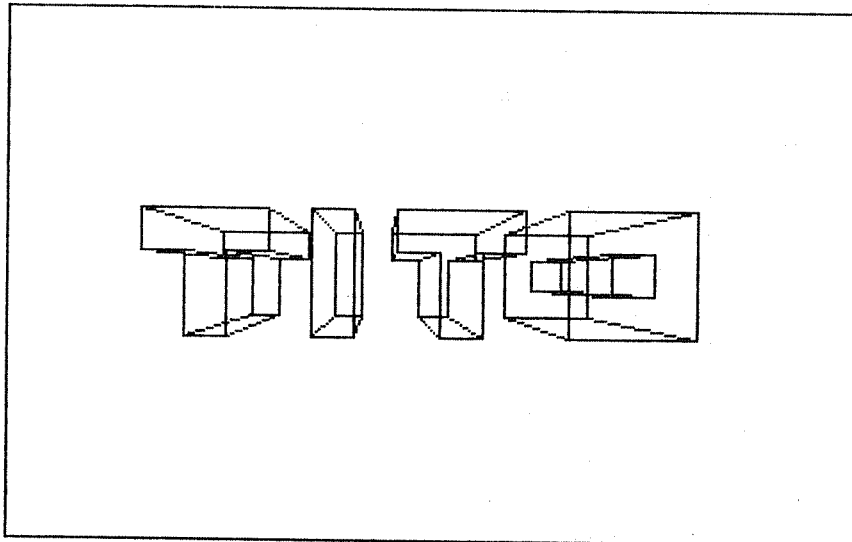


Fig. 11.66

Notez que seules les valeurs inférieures au Z minimum (après rotations) peuvent être utilisées comme points de fuite, à moins d'induire des déformations. Des points de fuite supérieurs au Z maximum produisent une inversion de la perspective, c'est-à-dire que les parties distantes ont l'air plus grandes que les parties proches. Des points de fuite entre le Z minimum et maximum produisent des images de l'objet complètement défor-

mées parce que le fond de l'objet est tracé à l'envers et les lignes qui relient les points de devant et de derrière se croisent au point de fuite.

Il y a une grande différence entre calculer la perspective sur des points avant et après qu'ils aient été soumis aux sous-programmes de rotations. Le programme 11.35 faisait les calculs avant les rotations. Ceci signifie que si par exemple les lignes donnent l'impression de converger vers un point de fuite derrière l'écran avant les rotations, une rotation de 90 degrés autour de l'axe des X fera que la perspective changera de direction et les lignes convergeront vers un point de fuite situé vers le bas de l'écran. La figure 11.67 montre le graphique produit par le programme 11.35, avec une rotation de 90 degrés autour de l'axe des X.

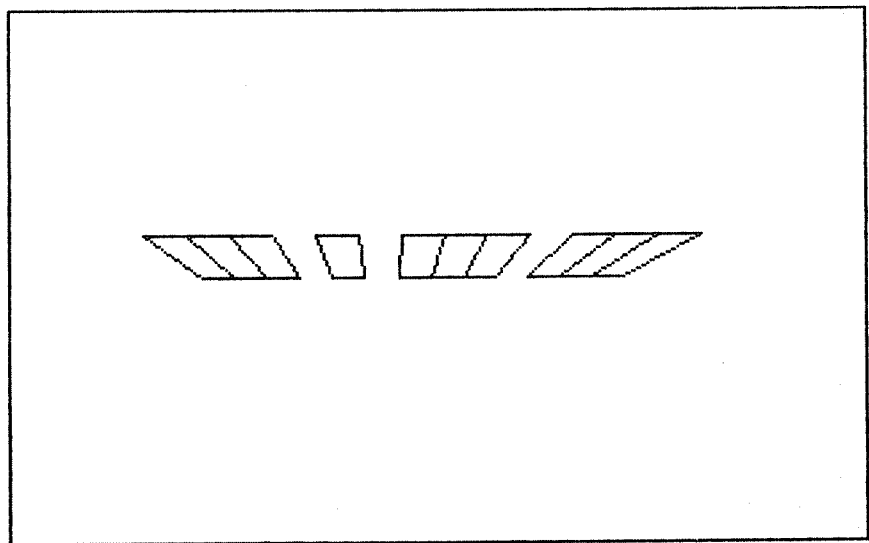


Fig. 11.67

Le fait que les lignes convergentes tournent également peut conduire à des représentations erronées, comme c'est le cas pour les rotations autour de l'axe des Y comprises entre 90 et -90 degrés, pour lesquelles la perspective est inversée comme cela arrive quand le point de fuite est plus grand que le Z maximum.

Une autre méthode consiste à appliquer la perspective après rotation. Le programme 11.36 accomplit les rotations avant de calculer la perspective au sous-programme 2000. La figure 11.68 montre « TITO » avec la même rotation que dans la figure 11.67. Notez que la perspective affecte toujours la profondeur, pas les lignes verticales.

```
0 '** 11-36 : Perspective after rot. **
10 F=1.745329E-02
15 NP=56:NL=84
20 SCREEN 1:CLS
```

```

30 INPUT "Rotation angles (around X,Y) ";B,C
40 INPUT "Vanishing point ";VANISHING.POINT:CLS
50 DIM X(60),Y(60),Z(60),LO(100),LD(100)
60 FOR I=1 TO NP:
    READ X(I),Y(I),Z(I):
    X(I)=X(I)-90:Y(I)=Y(I)+23:
NEXT
70 FOR I=1 TO NL:
    READ LO(I),LD(I):
NEXT
80 MAXIMUM.Z=-10000
90 FOR I=1 TO NP
110 IF Z(I)>MAXIMUM.Z
    THEN
        MAXIMUM.Z=Z(I)
120 NEXT
130 DISTANCE=MAXIMUM.Z-VANISHING.POINT
140 FOR I=1 TO NL
150 X=X(LO(I)):Y=Y(LO(I)):Z=Z(LO(I))
155 GOSUB 1000:GOSUB 2000
160 XT=X2:YT=Y2
170 X=X(LD(I)):Y=Y(LD(I)):Z=Z(LD(I))
175 GOSUB 1000:GOSUB 2000
180 LINE(140+XT,100+YT)-(140+X2,100+Y2)
190 NEXT
200 END
1000 '** Rotation subroutine **
1010 CB=COS(B*F):SB=SIN(B*F)
1020 CC=COS(C*F):SC=SIN(C*F)
1030 X1=X*CB+Z*SB
1040 Y1=Y
1050 Z1=-X*SB+Z*CB
1060 X2=X1
1070 Y2=Y1*CC-Z1*SC
1080 Z2=Y1*SC+Z1*CC
1090 RETURN
2000 '** Perspective subroutine **
2010 FACTOR=(Z2-VANISHING.POINT)/DISTANCE
2020 X2=X2*FACTOR:Y2=Y2*FACTOR
2030 RETURN
5000 DATA 0,-48,0,0,-32,0,16,-32,0,16,0,
    0,32,0,0,32,-32,0,48,-32,0,48,
    -48,0,0,-48,16,0,-32,16
5010 DATA 16,-32,16,16,0,16,32,0,16,32,
    -32,16,48,-32,16,48,-48,16,64,
    -48,0,64,0,0,80,0,0,80,-48,0
5020 DATA 64,-48,16,64,0,16,80,0,16,80,
    -48,16,96,-48,0,96,-32,0,112,
    -32,0,112,0,0,128,0,0,128,-32,0
5030 DATA 144,-32,0,144,-48,0,96,-48,16,
    96,-32,16,112,-32,16,112,0,16,
    128,0,16,128,-32,16,144,-32,16,
    144,-48,16
5040 DATA 160,-48,0,160,0,0,208,0,0,208,-48,
    0,176,-32,0,176,-16,0,192,-16,0,
    192,-32,0,160,-48,16,160,0,16
5050 DATA 208,0,16,208,-48,16,176,-32,16,
    176,-16,16,192,-16,16,192,-32,16
5060 DATA 1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,1
5070 DATA 9,10,10,11,11,12,12,13,13,
    14,14,15,15,16,16,9

```

```

5080 DATA 1,9,2,10,3,11,4,12,5,13,6,
          14,7,15,8,16
5090 DATA 17,18,18,19,19,20,20,17
5100 DATA 21,22,22,23,23,24,24,21
5110 DATA 17,21,18,22,19,23,20,24
5120 DATA 25,26,26,27,27,28,28,29,
          29,30,30,31,31,32,32,25
5130 DATA 33,34,34,35,35,36,36,37,
          37,38,38,39,39,40,40,33
5140 DATA 25,33,26,34,27,35,28,36,
          29,37,30,38,31,39,32,40
5150 DATA 41,42,42,43,43,44,44,41,
          49,50,50,51,51,52,52,49
5160 DATA 45,46,46,47,47,48,48,45,
          53,54,54,55,55,56,56,53
5170 DATA 41,49,42,50,43,51,44,52,
          45,53,46,54,47,55,48,56

```

PROGRAMME 11.36

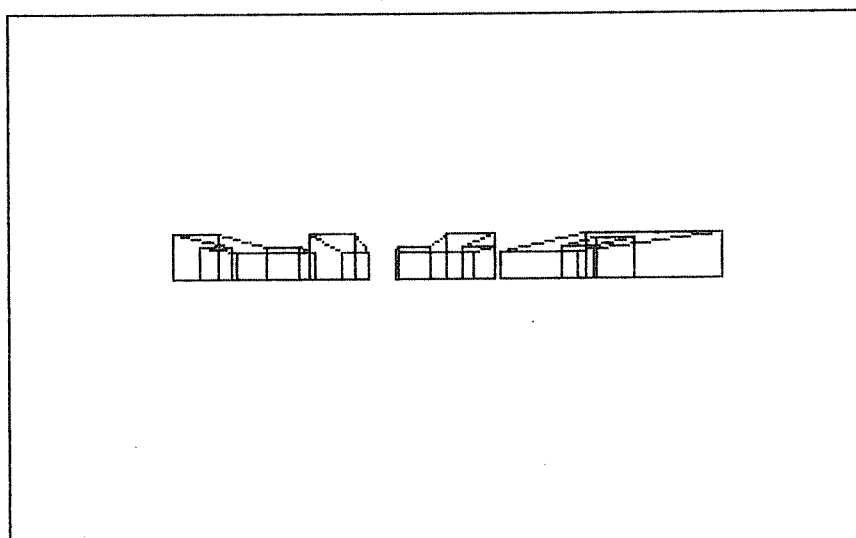


Fig. 11.68

Bien que la perspective ne soit pas impliquée, un problème peut surgir quand la plus petite valeur de Z tombe au-delà ou très près du point de fuite. Comme la figure produite par le programme 11.36 est plutôt plate (dans sa forme d'origine, le Z le plus petit est 0), on peut utiliser n'importe quelle valeur négative comme point de fuite. Toutefois, quand on fait tourner la figure, certaines coordonnées Z de ses points peuvent devenir négatives et tomber après le point de fuite, provoquant des déformations.

Quand le point de fuite se trouve à une distance considérable du Z minimum, on peut faire tourner l'objet en toute sécurité sur n'importe quelle position. Quand la rotation enmène certains points trop près

du point de fuite, ces parties peuvent être déformées. Dans la figure 11.69, par exemple, le « O » produit par le programme 11.36 est déformé parce qu'il tombe trop près du point de fuite.

Pour être toujours sûr, évitez les perspectives extrêmes, ou vérifiez que la plus petite valeur prise par la coordonnée Z de l'objet est relativement éloignée du point de fuite. Nous préférons calculer les perspectives après les rotations.

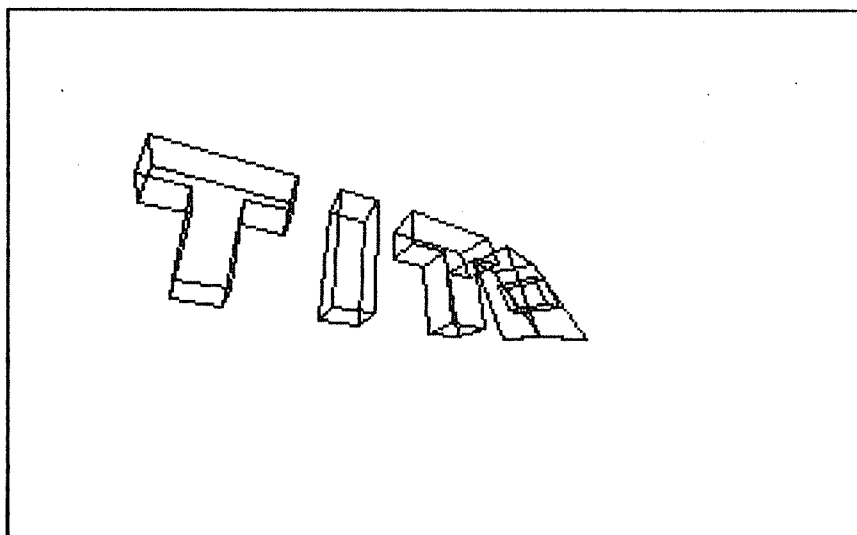


Fig. 11.69

Perspective horizontale. Outre la perspective en profondeur, il existe deux autres types de perspective liés à l'existence des points de fuite latéraux. La figure 11.70 montre l'effet d'introduire la perspective (et donc un second point de fuite) aux arêtes d'un cube parallèles à l'axe des X.

Le point de fuite est situé à droite du centre du graphique et par conséquent les lignes qui devraient normalement être parallèles à l'axe des X, convergent vers ce nouveau point de fuite latéral. Pour réaliser cet effet, la dimension verticale doit être petite pour les points situés près du point de fuite des X, et grande pour ceux qui en sont éloignés. Nous utiliserons ici la même technique que pour la perspective en profondeur : étant donné un point de fuite X et le point le plus éloigné de celui-ci (que le programme doit calculer à chaque fois parce qu'il change avec les rotations) dans le sens des X (l'équivalent du point le plus proche de l'observateur dans le sens des Z), le facteur d'échelle sera 0 pour les points situés au point de fuite des X, et 1 pour ceux situés au point le plus éloigné du point de fuite. Notez que lorsqu'on avait affaire au point de fuite des Z, les dimensions relatives X et Y des objets étaient toutes les

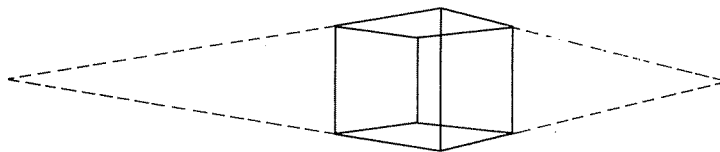


Fig. 11.70

deux affectées par leur distance au point de fuite. Ce n'est pas le cas avec la perspective basée sur le point de fuite X où la hauteur (Y) et la profondeur (Z) sont affectées, mais comme la coordonnée Z est ignorée, seules les variations de Y doivent être prises en compte.

Les équations pour calculer ce type de perspective sont :

$$\text{DISTANCE.X} = \text{MAX.X} - \text{VANISHING.X}$$

$$\text{FACTOR.X} = (\text{X} - \text{VANISHING.X}) / \text{DISTANCE.X}$$

Le programme 11.37 utilise le point de fuite rentré à la ligne 90 pour créer une vue en perspective d'une maison. La figure 11.71 montre la perspective produite par un point de fuite X situé à 400 et un point de fuite Z à -200.

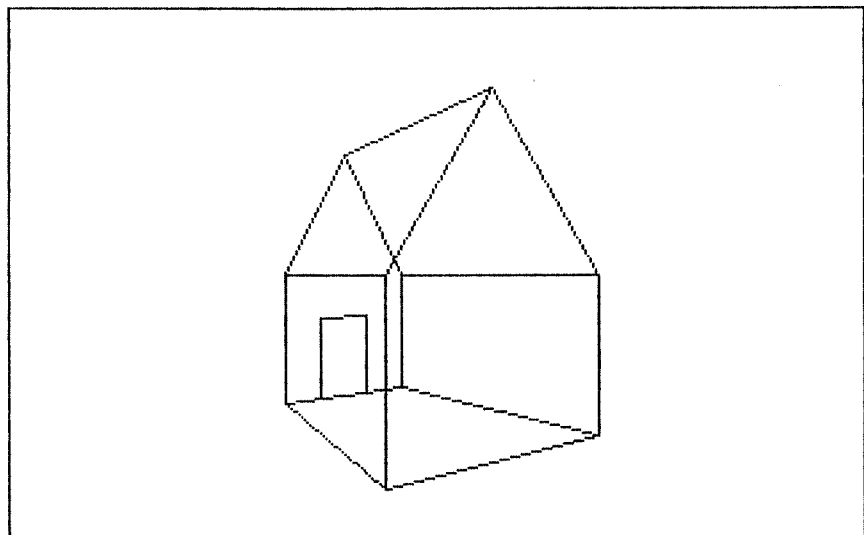


Fig. 11.71

```
0 '** 11-37 : Horizontal perspective **
10 F=1.745329E-02
20 FAC=40
30 NP=14:NL=18
40 SCREEN 1:CLS
```

```

50 INPUT "Rotation angles (X,Y,Z) ",RHO,PHI,THETA
60 SIN.THETA=SIN(THETA*F):COS.THETA=COS(THETA*F)
70 SIN.RHO=SIN(RHO*F):COS.RHO=COS(RHO*F)
80 SIN.PHI=SIN(PHI*F):COS.PHI=COS(PHI*F)
90 INPUT "Vanishing point (X,Z) ";
   VANISH.X,VANISH.Z:CLS
100 DIM X(60),Y(60),Z(60),LO(100),LD(100)
110 FOR I=1 TO NP:
   READ X(I),Y(I),Z(I):X(I)=(X(I)+3)*FAC:
   Y(I)=Y(I)*FAC:Z(I)=Z(I)*FAC:
NEXT
120 FOR I=1 TO NL:
   READ LO(I),LD(I):
NEXT
130 MAXIMUM.Z=-10000
140 FOR I=1 TO NP:
150   X=X(I):Y=Y(I):Z=Z(I):GOSUB 1000
160   IF Z3>MAXIMUM.Z
      THEN
        MAXIMUM.Z=Z3
170 NEXT
180 IF VANISH.X>0
   THEN
     FX=2
   ELSE
     FX=1
190 ON FX GOTO 200,210
200   MAXIMUM.X=-10000:GOTO 220
210   MAXIMUM.X=10000
220 FOR I=1 TO NP:
230   X=X(I):Y=Y(I):Z=Z(I):GOSUB 1000
240   ON FX GOTO 250,252
250   IF X3>MAXIMUM.X
      THEN
        MAXIMUM.X=X3
251   GOTO 253
252   IF X3<MAXIMUM.X
      THEN
        MAXIMUM.X=X3
253 NEXT
260 DISTANCE.Z=MAXIMUM.Z-VANISH.Z
270 DISTANCE.X=MAXIMUM.X-VANISH.X
280 FOR I=1 TO NL:
290   X=X(LO(I)):Y=Y(LO(I)):
   Z=Z(LO(I)):GOSUB 1000
300   GOSUB 2000:XT=X3:YT=Y3
310   X=X(LD(I)):Y=Y(LD(I)):
   Z=Z(LD(I)):GOSUB 1000
320   GOSUB 2000
330   LINE(60+XT,100+YT)-(60+X3,100+Y3)
340 NEXT
350 END
1000 'Calculate rotations
1010 X1=X*COS.RHO+Z*SIN.RHO
1020 Y1=Y
1030 Z1=-X*SIN.RHO+Z*COS.RHO
1040 X2=X1
1050 Y2=Y1*COS.PHI-Z1*SIN.PHI
1060 Z2=Y1*SIN.PHI+Z1*COS.PHI
1070 X3=X2*COS.THETA+Y2*SIN.THETA
1080 Y3=-X2*SIN.THETA+Y2*COS.THETA

```

```

1090 Z3=Z2
1100 RETURN
2000 '** Perspective subroutine **
2010 FACTOR.Z=(Z3-VANISH.Z)/DISTANCE.Z
2020 X3=X3*FACTOR.Z:Y3=Y3*FACTOR.Z
2030 FACTOR.X=(X3-VANISH.X)/DISTANCE.X
2040 Y3=Y3*FACTOR.X
2050 RETURN
5000 DATA -1,-1.5,-2,1,-1.5,-2,1,
        -1.5,0,0,-1.5,2,-1,-1.5,0
5010 DATA -1,1.5,-2,1,1.5,-2,1,1.5,
        0,0,1.5,2,-1,1.5,0
5020 DATA .4,-1.5,-2,.4,-1.5,
        -.7,-.4,-1.5,-.7,-.4,-1.5,-2
5030 DATA 1,2,2,3,3,4,4,5,5,1,6,7,7,8,
        8,9,9,10,10,6
5040 DATA 1,6,2,7,3,8,4,9,5,10,11,
        12,12,13,13,14

```

PROGRAMME 11.37

La figure 11.72 montre la même maison avec le point de fuite X à 32000 et le point de fuite Z à -300.

Perspective verticale. Le troisième type de perspective qui affecte la dimension relative des objets est la convergence de lignes verticales en un point situé en dessus ou en dessous de l'écran. La figure 11.73 montre un cube avec une telle déformation.

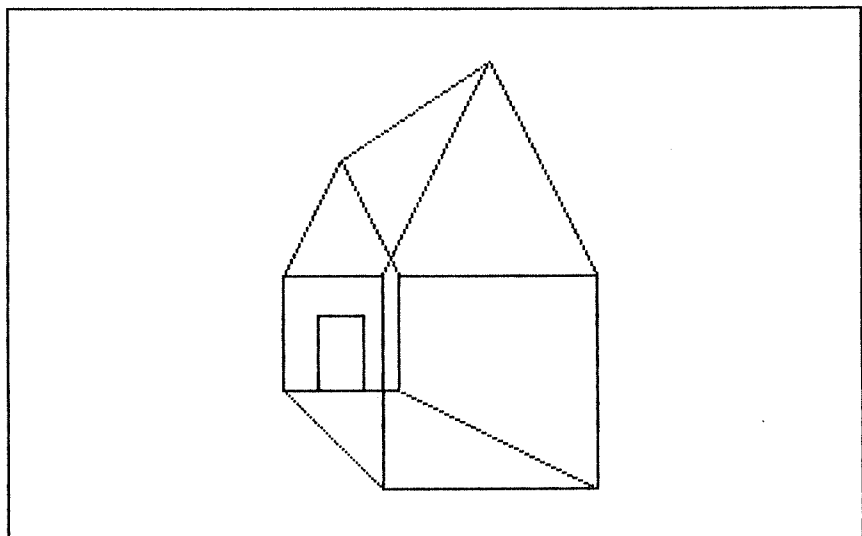


Fig. 11.72

Encore une fois un point de fuite (cette fois dans le sens des Y) sera nécessaire pour les calculs. A la différence de la perspective X qui affecte

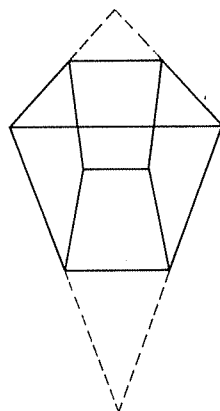


Fig. 11.73

la dimension relative des composantes verticales, cette nouvelle perspective affecte la dimension des composantes horizontales des objets. (Elle affecte aussi la profondeur mais comme nous ne l'utilisons pas dans le dessin final, nous pouvons l'ignorer.) Une fois le point de fuite et son point le plus éloigné trouvés, le facteur d'échelle affecte la coordonnée X de chaque point. Les équations pour calculer les facteurs d'échelle verticaux sont :

$$\text{DISTANCE.Y} = \text{MAX.Y} - \text{VANISHING.Y}$$

$$\text{FACTOR.Y} = (\text{Y} - \text{VANISHING.Y}) / \text{DISTANCE.Y}$$

Le programme 11.38 rentre les trois points de fuite et dessine une maison avec les transformations de perspective correspondantes.

```

0  '** 11-38 : 3-axes perspective **
10 F=1.745329E-02
20 NP=14:NL=18
30 SCREEN 1:CLS
40 INPUT"Rotation angles (X,Y,Z) ",RHO,PHI,THETA
50 SIN.THETA=SIN(THETA*F):COS.THETA=COS(THETA*F)
60 SIN.RHO=SIN(RHO*F):COS.RHO=COS(RHO*F)
70 SIN.PHI=SIN(PHI*F):COS.PHI=COS(PHI*F)
80 INPUT"Vanishing points (X,Y,Z) ";
   VANISH.X,VANISH.Y,VANISH.Z
90 CLS
100 DIM X(60),Y(60),Z(60),LO(100),LD(100)
110 FOR I=1 TO NP:
   READ X(I),Y(I),Z(I):X(I)=1.7*(X(I)-120):
   Y(I)=1*(Y(I)+60):Z(I)=1*Z(I):
NEXT
120 FOR I=1 TO NL:
   READ LO(I),LD(I):
NEXT
125 ' ** Find maximum Z **

```

```

130 MAXIMUM.Z=-10000
140 FOR I=1 TO NP
150   X=X(I):Y=Y(I):Z=Z(I):GOSUB 1000
160   IF Z3>MAXIMUM.Z
      THEN
        MAXIMUM.Z=Z3
170 NEXT
175 '** Find maximum X **
180 IF VANISH.X>0
      THEN
        FX=2 ELSE FX=1
190 ON FX GOTO 200,210
200   MAXIMUM.X=-10000:GOTO 220
210   MAXIMUM.X=10000
220 FOR I=1 TO NP
230   X=X(I):Y=Y(I):Z=Z(I):GOSUB 1000
240   ON FX GOTO 250,270
250   IF X3>MAXIMUM.X
      THEN
        MAXIMUM.X=X3
260   GOTO 280
270   IF X3<MAXIMUM.X
      THEN
        MAXIMUM.X=X3
280 NEXT
285 '** Find maximum Y **
290 IF VANISH.Y>0
      THEN
        FY=2
      ELSE
        FY=1
300 ON FY GOTO 310,320
310   MAXIMUM.Y=-10000:GOTO 330
320   MAXIMUM.Y=10000
330 FOR I=1 TO NP
340   X=X(I):Y=Y(I):Z=Z(I):GOSUB 1000
350   ON FY GOTO 360,380
360   IF Y3>MAXIMUM.Y
      THEN
        MAXIMUM.Y=Y3
370   GOTO 390
380   IF Y3<MAXIMUM.Y
      THEN
        MAXIMUM.Y=Y3
390 NEXT
400 DISTANCE.Z=MAXIMUM.Z-VANISH.Z
410 DISTANCE.Y=ABS(MAXIMUM.Y-VANISH.Y)
420 DISTANCE.X=MAXIMUM.X-VANISH.X
430 FOR I=1 TO NL
440   X=X(LD(I)):Y=Y(LD(I)):
      Z=Z(LD(I)):GOSUB 1000
450   GOSUB 2000:XT=X3:YT=Y3
460   X=X(LD(I)):Y=Y(LD(I)):
      Z=Z(LD(I)):GOSUB 1000
470   GOSUB 2000
480   LINE(160+XT,40+YT)-(160+X3,40+Y3)
490 NEXT
500 W$=INPUT$(1):END
1000 '** Rotation subroutine **
1010 X1=X*COS.RHO+Z*SIN.RHO
1020 Y1=Y

```

```

1030 Z1=-X*SIN.RHO+Z*COS.RHO
1040 X2=X1
1050 Y2=Y1*COS.PHI-Z1*SIN.PHI
1060 Z2=Y1*SIN.PHI+Z1*COS.PHI
1070 X3=X2*COS.THETA+Y2*SIN.THETA
1080 Y3=-X2*SIN.THETA+Y2*COS.THETA
1090 Z3=Z2
1100 RETURN
2000 '** Adjust perspective **'
2010 FACTOR.Z=(Z3-VANISH.Z)/DISTANCE.Z
2020 X3=X3*FACTOR.Z:Y3=Y3*FACTOR.Z
2030 FACTOR.X=(X3-VANISH.X)/DISTANCE.X
2040 Y3=Y3*FACTOR.X
2050 FACTOR.Y=(Y3-VANISH.Y)/DISTANCE.Y
2060 X3=X3*FACTOR.Y
2070 RETURN
5000 DATA 80,80,-60,160,80,-60,160,0,-60,
          120,-80,-60,80,0,-60,80,80,60,
          160,80,60,160,0,60,120,-80,60,
          80,0,60,136,80,-60,136,28,-60,
          104,28,-60,104,80,-60
5010 DATA 1,2,2,3,3,4,4,5,5,1,6,7,7,
          8,8,9,9,10,10,6
5020 DATA 1,6,2,7,3,8,4,9,5,10,11,
          12,12,13,13,14

```

PROGRAMME 11.38

La figure 11.74 montre l'effet de points de fuite situés à $X = 32000$, $Y = 500$ et $Z = -500$.

La figure 11.75 montre une maison avec les trois perspectives combinées.

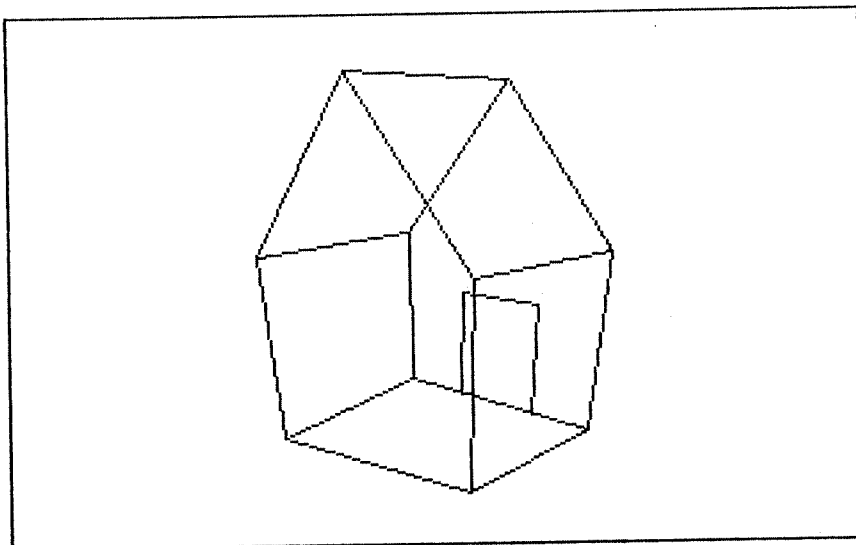


Fig. 11.74

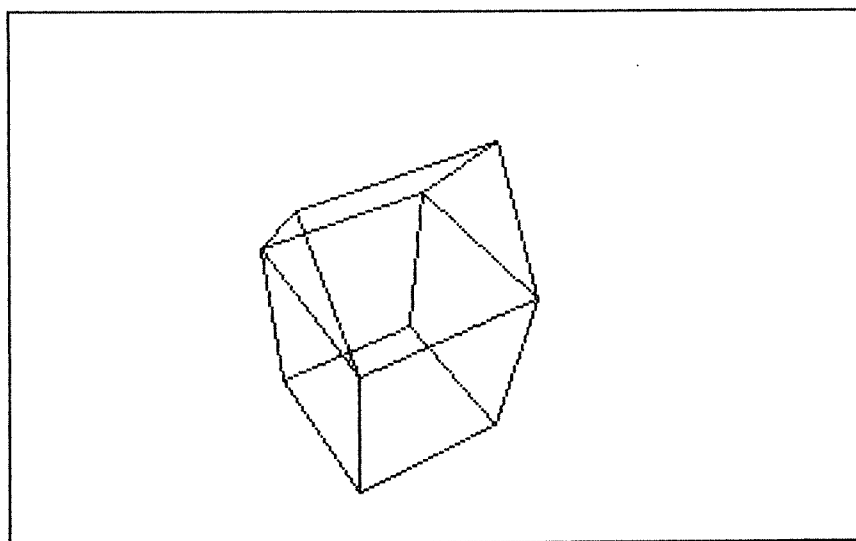


Fig. 11.75

11.2.5 Les translations

Un objet en 3-D peut être transféré sur l'écran en ajoutant des valeurs aux coordonnées de chaque point de la même manière que nous l'avons fait au paragraphe 2.7. Ces valeurs peuvent être ajoutées avant ou après avoir calculé les rotations et les transformations de perspective, et le moment où se font ces additions peut provoquer une grande différence dans l'image finale.

Quand on effectue les translations après les transformations (perspective et rotations), le graphique est déplacé à un endroit différent de l'écran, mais sans être modifié dans son aspect ³. En ajoutant des valeurs aux coordonnées X de chaque point, on déplacera l'ensemble du graphique sur la droite (ou la gauche si on utilise une valeur négative) et des valeurs ajoutées à la coordonnée Y déplaceront la figure vers le haut (ou vers le bas si on prend une valeur négative). Changer la coordonnée Z est complètement inutile ici, vu qu'elle est ignorée.

Les translations faites avant les transformations peuvent avoir un effet significatif sur les figures résultantes.

Effet des translations sur les rotations. Supposez que l'objet que l'on considère soit une maison située de telle manière que ses murs soient plus ou moins équidistants du centre du système (le point (0,0,0)). Une rotation autour de l'axe des X change l'orientation, mais la maison reste proche de sa position d'origine, comme on le voit sur la figure 11.76.

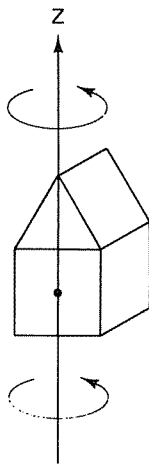


Fig. 11.76

Si l'objet est maintenant transféré à un nouvel emplacement en ajoutant par exemple une valeur aux coordonnées X de ses points, une rotation ultérieure autour de l'axe des X aura deux effets : la maison subira la même rotation que sans translation, mais elle sera aussi transférée sur un chemin circulaire, comme le montre la figure 11.77.

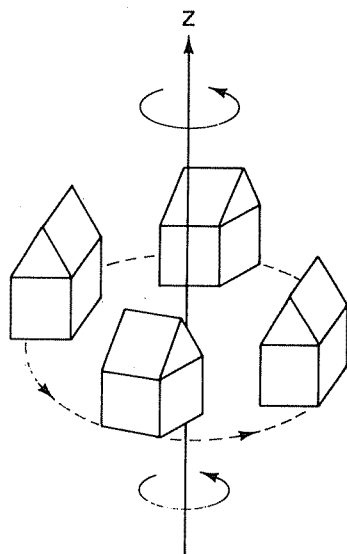


Fig. 11.77

Du fait de la rotation, une nouvelle translation peut être nécessaire ensuite pour ramener l'objet sur l'écran. Ceci s'applique de même aux

translations le long de l'axe des Y ou des Z comme le montrent les figures 11.78 et 11.79.

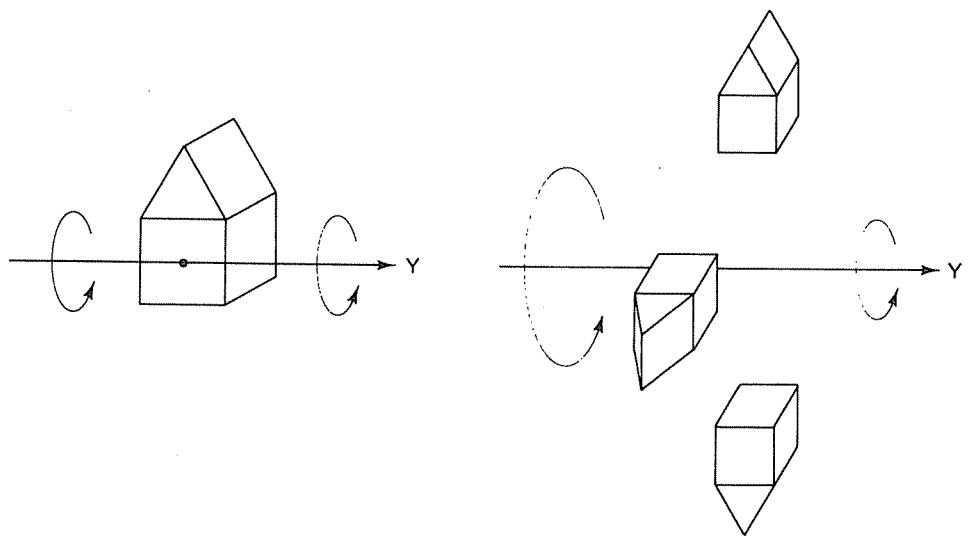


Fig. 11.78

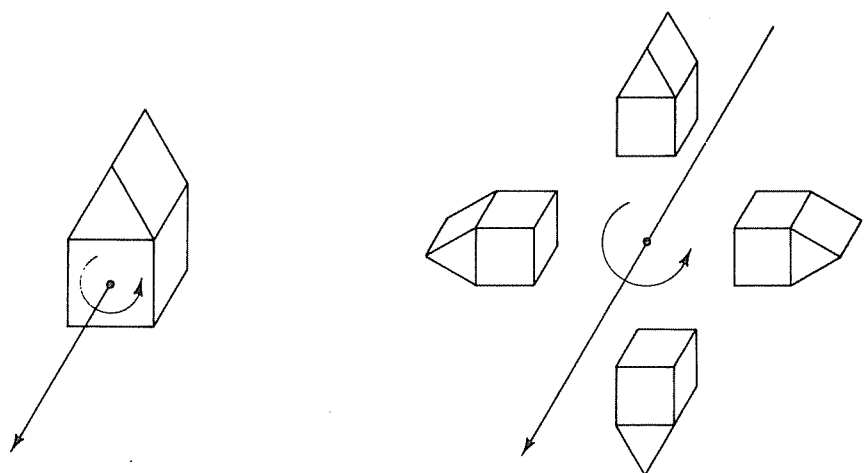


Fig. 11.79

Effets des translations sur la perspective. Les translations faites avant les calculs de perspective peuvent être de deux types. Nous allons commencer par étudier l'effet des translations des coordonnées X et Y, et ensuite celles des coordonnées Z relatives à la profondeur.

La figure 11.80 montre une maison vue de face, c'est-à-dire avec le point de fuite situé exactement derrière elle.

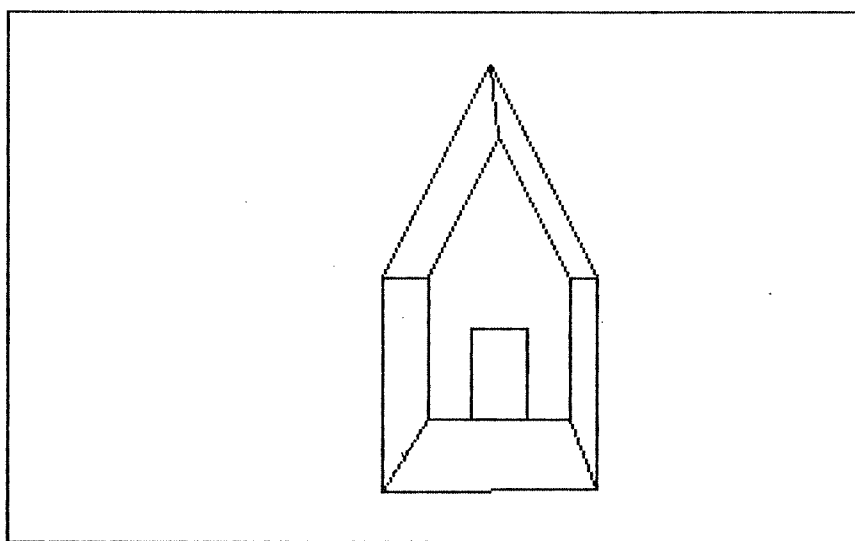


Fig. 11.80

Si on ajoute une valeur à la coordonnée X de chaque point et que les calculs de perspective sont alors effectués, la maison sera vue d'un côté, comme le montre la figure 11.81. Notez que les lignes de profondeur convergent vers un point situé sur la gauche de la maison.

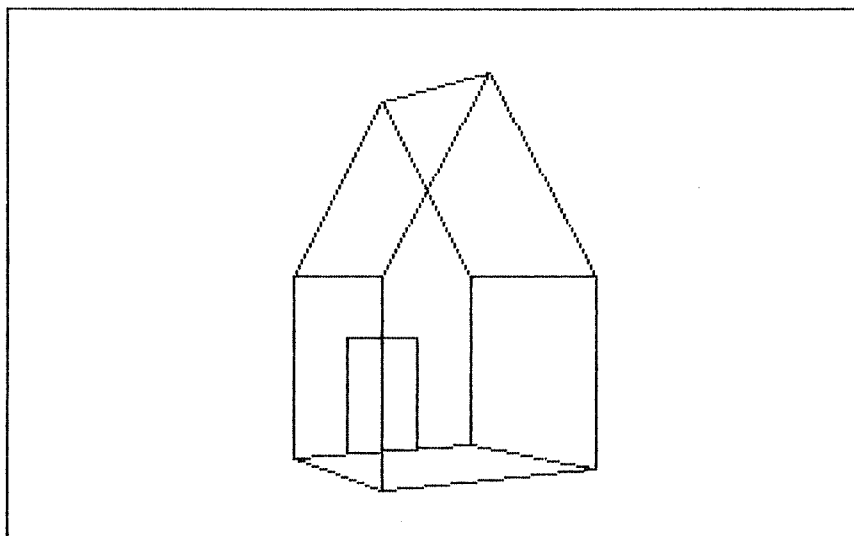


Fig. 11.81

La figure 11.82 montre l'effet de la translation de la maison sur la gauche avant application de la perspective. On peut s'attendre à un effet semblable avec une translation dans le sens des Y .

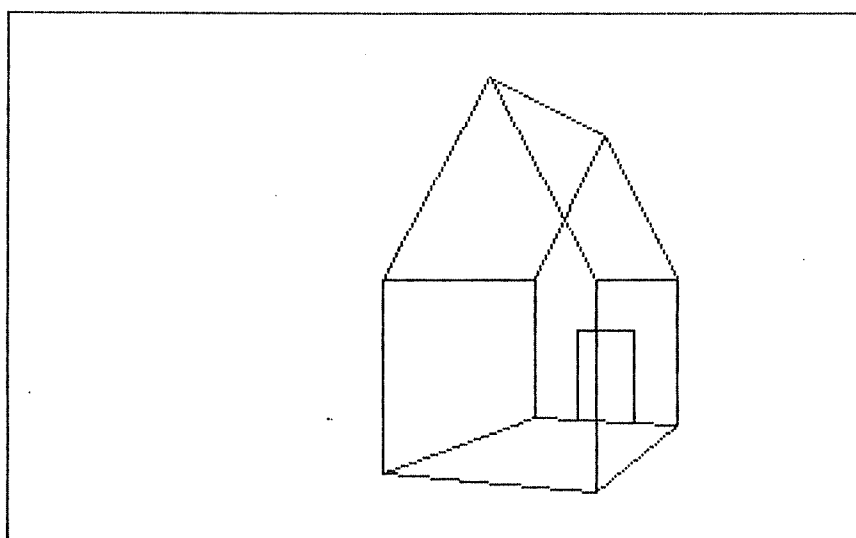


Fig. 11.82

Une translation dans le sens des Z a un effet complètement différent sur la perspective en profondeur. Si le point de fuite était derrière la maison avant la translation, il restera là après, mais la maison aura l'air plus éloignée (ou plus proche) de l'observateur. Selon le point de fuite, la translation peut avoir un effet plus ou moins grand : quand le point de fuite est (proportionnellement) proche de l'objet, un petit changement dans la coordonnée Z fera bouger l'objet considérablement, tandis qu'un point de fuite distant amènera de faibles modifications de perspective avec de petites translations dans le sens des Z .

Ces changements s'appliquent aussi à la perspective latérale : les lignes convergentes parallèles à l'axe des X seront affectées par des translations des groupes (Y,Z) , et X . Les lignes convergentes dans le sens des Y seront affectées par les groupes (X,Z) , et (Y) .

11.2.6 Echelle et distorsion

Pour mettre à l'échelle un objet tridimensionnel, les coordonnées de tous ses points doivent être multipliées par un facteur commun. Comme dans le cas des translations, le moment où intervient cette opération (avant ou après les transformations) conduit à des différences de résultat importantes.

Quand la mise à l'échelle est faite après que l'image ait été traitée, les proportions, la perspective et tout le reste sont préservés, parce que cette mise à l'échelle est parfaitement équivalente à celle d'une image bidimensionnelle. La mise à l'échelle des coordonnées Z n'a pas d'effet ici).

Mettre à l'échelle avant les transformations est un autre problème quand il s'agit de perspective. (Les rotations ne sont pas affectées par la mise à l'échelle.) Du fait que le rapport de la distance maximum (la distance entre le point de coordonnée Z maximum et le point de fuite) est modifié, la perspective peut être grandement affectée. Si tous les points restent d'un seul côté du point de fuite, on peut dire qu'une augmentation (ou diminution) de la dimension de l'objet aura (du point de vue de la perspective) le même effet qu'elle aurait eu dans la réalité : une maison normale a une perspective normale, alors que les côtés d'une maison gigantesque de plusieurs kilomètres de hauteur sembleront converger vers l'infini. Si la multiplication donne des valeurs qui tombent de l'autre côté d'un des points de fuite, de grands dégâts s'ensuivent (à moins que des images déformées soient ce que vous recherchez).

Le programme 11.39 dessine les lettres de la figure 11.40 en effectuant deux types de changement d'échelle : à la ligne 62 toutes les coordonnées (X, Y et Z) sont multipliées par le facteur rentré à la ligne 35, et à la ligne 180 les points sont réduits (les valeurs sont divisées par SCALE). Si la perspective n'était pas impliquée, cette double opération s'annulerait. Toutefois, dans le cas présent, l'augmentation (ou la diminution) des coordonnées Z a un effet sur la perspective.

```

0 '* 11-39 : Scaling with perspective *
1 DEF FN A$(X)=STR$(CINT(X))
10 F=1.745329E-02
15 NP=56:NL=84
20 SCREEN 1:COLOR 1,0:CLS
35 INPUT "Multiply by: ";SCALE:
   LINE(0,0)-(319,199),2,BF
40 VANISHING.POINT=-100
50 DIM X(60),Y(60),Z(60),LD(100),LD(100)
60 FOR I=1 TO NP:
   READ X(I),Y(I),Z(I)
62   X(I)=X(I)*SCALE:Y(I)=Y(I)*SCALE:
   Z(I)=Z(I)*SCALE:
   NEXT
70 FOR I=1 TO NL:
   READ LD(I),LD(I):
   NEXT
80 MAXIMUM.Z=-10000
90 FOR I=1 TO NP
110   IF Z(I)>MAXIMUM.Z
   THEN
     MAXIMUM.Z=Z(I)
120 NEXT
130 DISTANCE=MAXIMUM.Z-VANISHING.POINT
140 FOR I=1 TO NL
150   X=X(LD(I)):Y=Y(LD(I)):Z=Z(LD(I))

```

```

155 GOSUB 2000
160 XT=X:YT=Y
170 X=X(LD(I)):Y=Y(LD(I)):Z=Z(LD(I))
175 GOSUB 2000
180 LINE(140+XT/SCALE,100+YT/SCALE)-
      (140+X/SCALE,100+Y/SCALE)
190 NEXT:PAINT(0,0),0,3
200 END
2000 ** Adjust perspective **
2010 FACTOR=(Z-VANISHING.POINT)/DISTANCE
2020 X=X*FACTOR:Y=Y*FACTOR
2030 RETURN
5000 DATA -90,-25,0,-90,-9,0,-74,-9,0,-74,
      23,0,-58,23,0,-58,-9,0,-42,-9,0,
      -42,-25,0,-90,-25,16,-90,-9,16,
      -74,-9,16,-74,23,16,-58,23,16,
      -58,-9,16,-42,-9,16
5010 DATA -42,-25,16,-26,-25,0,-26,23,0,-10,
      23,0,-10,-25,0,-26,-25,16,-26,23,
      16,-10,23,16,-10,-25,16,6,-25,0,
      6,-9,0,22,-9,0,22,23,0,38,23,0
5020 DATA 38,-9,0,54,-9,0,54,-25,0,6,-25,16,
      6,-9,16,22,-9,16,22,23,16,38,23,
      16,38,-9,16,54,-9,16,54,-25,16,
      70,-25,0,70,23,0,118,23,0
5030 DATA 118,-25,0,86,-9,0,86,7,0,102,7,
      0,102,-9,0,70,-25,16,70,23,16,118,
      23,16,118,-25,16,86,-9,16,
      86,7,16,102,7,16,102,-9,16
5040 DATA 1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,1
5050 DATA 9,10,10,11,11,12,12,13,13,
      14,14,15,15,16,16,9
5060 DATA 1,9,2,10,3,11,4,12,5,13,6,
      14,7,15,8,16
5070 DATA 17,18,18,19,19,20,20,17
5080 DATA 21,22,22,23,23,24,24,21
5090 DATA 17,21,18,22,19,23,20,24
5100 DATA 25,26,26,27,27,28,28,29,29,
      30,30,31,31,32,32,25
5110 DATA 33,34,34,35,35,36,36,37,37,
      38,38,39,39,40,40,33
5120 DATA 25,33,26,34,27,35,28,36,29,
      37,30,38,31,39,32,40
5130 DATA 41,42,42,43,43,44,44,41,49,
      50,50,51,51,52,52,49
5140 DATA 45,46,46,47,47,48,48,45,53,
      54,54,55,55,56,56,53
5150 DATA 41,49,42,50,43,51,44,52,45,
      53,46,54,47,55,48,56

```

PROGRAMME 11.39

Cette manière de changer la perspective de la profondeur a un côté intéressant : comme le point de fuite n'est pas changé et que seules les coordonnées Z sont affectées, c'est vraiment le Z maximum (et donc la distance du point de fuite) qui varie. Si toutes les coordonnées Z d'un objet sont positives et que le point de fuite est négatif, cette double opération

garantit que la perspective ne produira jamais d'autres types de déformations (comme celles produites par un point de fuite supérieur à certains Z) parce qu'une valeur positive multipliée par n'importe quel facteur positif produira toujours des valeurs positives. Essayez le programme 11.39 avec des facteurs d'échelle grands et petits.

Si toutes les coordonnées d'un objet ne sont pas multipliées par un facteur commun, l'objet est déformé. Si par exemple, les coordonnées X sont multipliées par 2, l'objet aura l'air élargi dans la direction horizontale. Le programme 11.40 dessine la fusée de la figure 11.83. Les données (DATA) de la ligne 300 correspondent aux 17 points, et celles des lignes 350 à 390 aux 28 lignes.

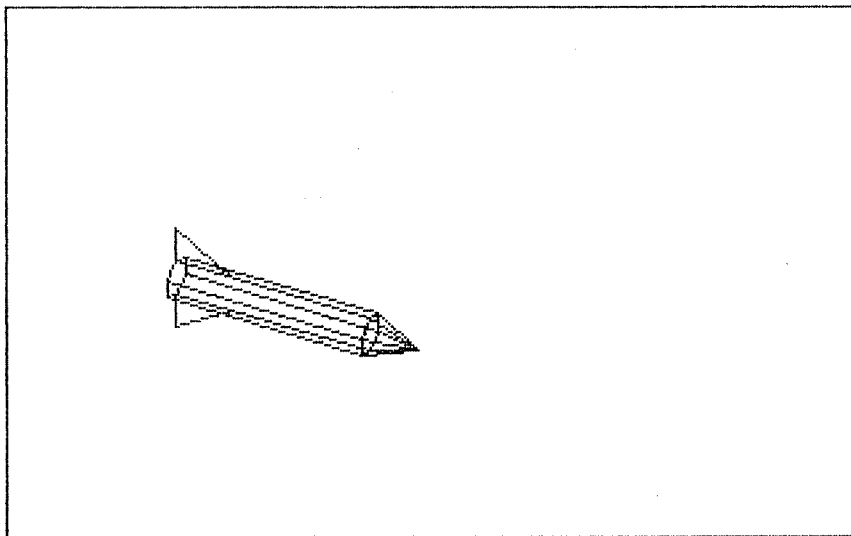


Fig. 11.83

```

0  ** 11-40 : Rocket **
5  F=1.745329E-02
10 SCREEN 1:CLS
12 B=25:C=40
20 DIM X(50),Y(50),Z(50),LO(50),LD(50)
30 FOR I=1 TO 17:
    READ X(I),Y(I),Z(I):
  NEXT
40 FOR I=1 TO 28:
    READ LO(I),LD(I):
  NEXT
50 FOR I=1 TO 28
60  X=4*X(LO(I)):Y=4*Y(LO(I)):
    Z=4*Z(LO(I)):GOSUB 1000
70  XT=X2:YT=Y2
80  X=4*X(LD(I)):Y=4*Y(LD(I)):
    Z=4*Z(LD(I)):GOSUB 1000
90  LINE(60+XT,100+YT)-(60+X2,100+Y2)

```

```

100 NEXT
110 END
300 DATA 0,0,2,0,1,0,0,3,0,0,4,2,0,
        3,4,0,1,4
310 DATA 20,0,2,20,1,0,20,3,0,20,4,
        2,20,3,4,20,1,4
320 DATA 25,2,2,0,-4,2,0,8,2,6,0,2,6,4,2
350 DATA 1,2,2,3,3,4,4,5,5,6,6,1
360 DATA 7,8,8,9,9,10,10,11,11,12,12,7
370 DATA 1,7,2,8,3,9,4,10,5,11,6,12
380 DATA 7,13,8,13,9,13,10,13,11,13,12,13
390 DATA 1,14,14,16,4,15,15,17
1000 '** Rotation subroutine **
1010 CB=COS(B*F):SB=SIN(B*F)
1020 CC=COS(C*F):SC=SIN(C*F)
1030 X1=X*CB+Z*SB
1040 Y1=Y
1050 Z1=-X*SB+Z*CB
1060 X2=X1
1070 Y2=Y1*CC-Z1*SC
1080 Z2=Y1*SC+Z1*CC
1090 RETURN

```

PROGRAMME 11.40

Si on change la ligne 30 en

```

30 FOR I=1 TO 17:
    READ X(I),Y(I),Z(I):
    X(I)=SCALE.X*X(I):
NEXT

```

et qu'on ajoute la ligne suivante :

```
15 INPUT "Scale in X ";SCALE.X
```

la longueur de la fusée sera changée, alors que la hauteur sera conservée comme le montre la figure 11.84.

Si on change la ligne 30 en

```

30 FOR I=1 TO 17:
    READ X(I),Y(I),Z(I):
    Y(I)=SCALE.Y*Y(I):
NEXT

```

et qu'on ajoute la ligne suivante au programme 11.40,

```
15 INPUT "Scale in Y ";SCALE.Y
```

la hauteur sera changée et la fusée aura l'air plus grosse (quand SCALE.Y est supérieur à 1) comme on le voit à la figure 11.85. La distorsion des coordonnées Z aura un effet sur la perspective de profondeur.

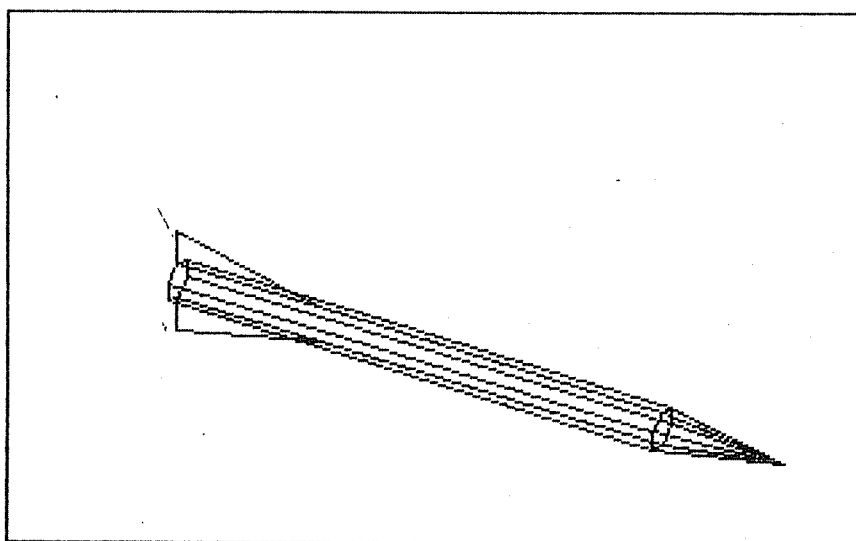


Fig. 11.84

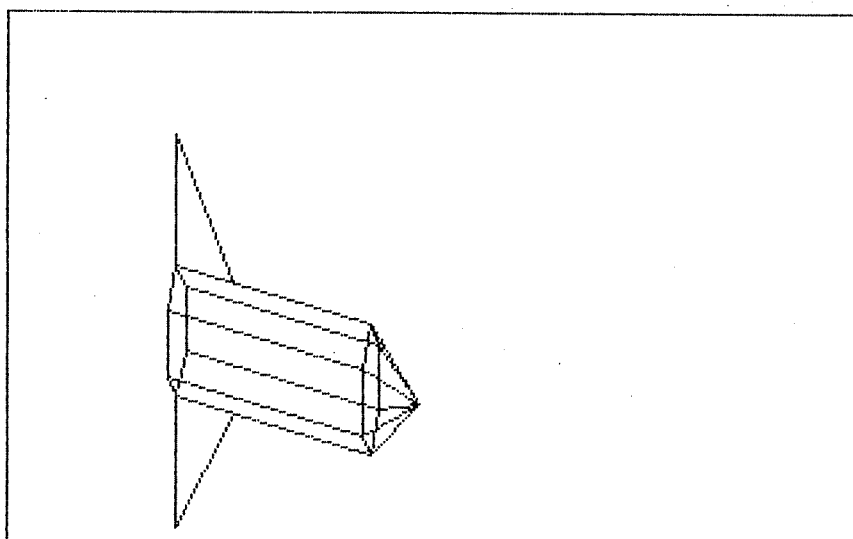


Fig. 11.85

11.2.7 Suppression des lignes cachées

Les représentations de tous les objets solides que nous avons vues jusqu'ici ressemblaient à des modèles en fil de fer. L'effet est très intéres-

sant et utile. Toutefois quand l'objet est composé de plusieurs surfaces et côtés, la projection n'est pas très nette et quelquefois même impossible à comprendre. Nous allons présenter maintenant une méthode pour effacer les lignes qui seraient invisibles dans la réalité.

A la différence de la représentation interne que nous avons utilisée jusqu'ici, nous assemblerons les objets à partir de polygones⁴, les faces qui constituent le solide. Nous prendrons une maison⁵ pour expliquer la méthode à travers un exemple. La figure 11.86 montre les polygones qui constituent l'objet.

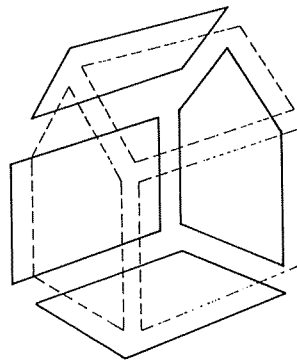


Fig. 11.86

Nous affectons à chaque polygone un centre, un point aussi proche de son centre de gravité qu'il soit possible de calculer et une ligne (que nous appellerons le « vecteur normal ») pointant en dehors de l'objet. La figure 11.87 montre les centres et les vecteurs normaux de trois des côtés.

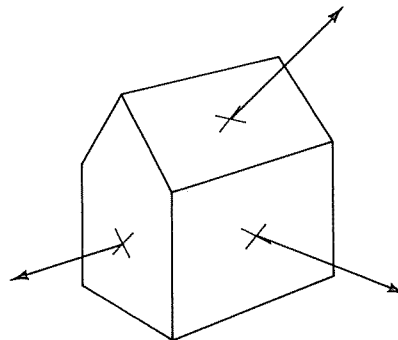


Fig. 11.87

Le processus pour dessiner la maison avec des murs solides est de trier les polygones en se guidant sur la position des vecteurs normaux. Quand

un polygone est dans le fond de l'objet, son vecteur normal pointera vers l'intérieur, dans le sens de l'axe $-Z$. Les polygones situés sur le devant de l'objet auront leurs vecteurs normaux qui pointeront à l'extérieur du plan $X-Y$, vers l'observateur, positivement suivant l'axe des Z . Une approche facile serait de ne pas dessiner les polygones dont les vecteurs normaux pointent dans le sens $-Z$. Bien que la méthode marcherait, elle ne permettrait pas l'utilisation de murs transparents (qu'on étudiera plus tard). Comme nous avons les coordonnées Z des deux points qui définissent chaque vecteur normal, nous pouvons trouver facilement l'orientation du vecteur en soustrayant une coordonnée Z de l'autre. Prenez un vecteur normal pointant dans le sens $-Z$. Le point dans le polygone a une coordonnée Z supérieure à celle du point extérieur à l'objet. Si le vecteur pointe dans le sens Z , le point à l'extérieur de l'objet aura une coordonnée Z plus grande.

Une fois les polygones triés, ils sont dessinés un par un, à partir de celui qui aura un vecteur normal pointant le plus dans le sens $-Z$ jusqu'à celui dont le vecteur pointe le plus dans le sens Z . Pour s'assurer que les faces de devant (qui seront dessinées en dernier) effacent les parties des faces arrière qui sont invisibles, nous utiliserons la méthode bestiale (voir paragraphe 4.3) : nous dessinerons chaque polygone deux fois, chaque fois d'une couleur différente.

Comme pour la représentation vectorielle, nous mémoriserons tous les points de l'objet dans des tableaux X , Y et Z et les données du polygone dans le tableau à trois dimensions M . Le premier index de M indiquera le numéro du polygone, le deuxième le numéro de ligne dans le polygone et le troisième indiquera si le point est l'origine ou la destination de la ligne. Finalement, la valeur $M(I, 0, 0)$ indiquera le nombre de lignes dans le polygone I , le point $M(I, 0, 1)$ indiquera le centre du polygone et $M(I, 0, 2)$ indiquera le point qui relié au centre du polygone forme le vecteur normal.

Les données des polygones qui composent la maison, leurs centres et les points pour créer les vecteurs normaux sont calculées aux lignes 40 à 140 du programme 11.41. Nous avons fixé arbitrairement à 6 le nombre maximum de côtés du polygone. Afin de déplacer chaque simple coordonnée de chaque polygone dans le processus de tri, nous utilisons un tri qui utilise des pointeurs sur les polygones, mais ne change pas la position des nombres en train d'être triés⁶. Les figures 11.88 et 11.89 montrent deux vues de la maison, maintenant avec des murs solides et non transparents. Les polygones qui composent les objets solides ne doivent pas intercepter d'autres polygones. La figure 11.90 montre deux polygones qui se coupent, notez qu'il importe peu de savoir quel polygone a été tracé le premier, le résultat ne correspondra pas à ce dont les choses auraient l'air dans la réalité.

```
0 *** 11-41 : Hidden line removal **
3 F=1.745329E-02
5 B=50
```

```

6 C=30
7 FAC=2.5
8 DX=150: DY=40
10 SCREEN 1:CLS
20 DIM X(100),Y(100),Z(100),
    M(50,6,2),TF(50),ZT(50)
22 FOR I=1 TO 50:
    TF(I)=I:
    NEXT
30 READ N
40 FOR I=1 TO N
50   READ X(I),Y(I),Z(I)
55   X(I)=FAC*X(I):
    Y(I)=FAC*Y(I):Z(I)=FAC*Z(I)
60 NEXT
70 READ NP
80 FOR I=1 TO NP
90   READ M(I,0,0)
100  FOR J=1 TO M(I,0,0)
110   READ M(I,J,1),M(I,J,2)
120  NEXT
130  READ M(I,0,1),M(I,0,2)
140  NEXT
150  FOR I=1 TO NP
155   P=M(I,0,1)
156   X=X(P):Y=Y(P):Z=Z(P)
157   GOSUB 1000
160   ZT(I)=Z2
165  NEXT
170  FOR I=1 TO NP-1
171   FOR J=I+1 TO NP
174    P=M(TF(I),0,2):X=X(P):Y=Y(P):Z=Z(P)
175    GOSUB 1000:ZA=Z2-ZT(TF(I))
176    P=M(TF(J),0,2):X=X(P):Y=Y(P):Z=Z(P)
177    GOSUB 1000:ZB=Z2-ZT(TF(J))
178    IF ZA>ZB
        THEN
            SWAP TF(I),TF(J)
185   NEXT
190  NEXT
300  FOR I=1 TO NP:II=TF(I)
310   FOR L=1 TO 1
320    FOR K=1 TO 1
330     FOR J=1 TO M(II,0,0)
340      P=M(II,J,1)
350      X=X(P):Y=Y(P):Z=Z(P)
355      GOSUB 1000:XA=X2:YA=Y2
360      P=M(II,J,2)
370      X=X(P):Y=Y(P):Z=Z(P)
375      GOSUB 1000:XB=X2:YB=Y2
380      LINE(XA+DX,YA+DY)-
        (XB+DX,YB+DY),K
390    NEXT
400    P=M(II,0,1):'center
410    X=X(P):Y=Y(P):Z=Z(P)
415    GOSUB 1000
420    PAINT(X2+DX,Y2+DY),K,K
430   NEXT
440  NEXT
450  FOR J=1 TO M(II,0,0)
460   P=M(II,J,1)

```



```

470      X=X(P):Y=Y(P):Z=Z(P)
475      GOSUB 1000:XA=X2:YA=Y2
480      P=M(II,J,2)
490      X=X(P):Y=Y(P):Z=Z(P)
495      GOSUB 1000:XB=X2:YB=Y2
500      LINE(XA+DX,YA+DY)-(XB+DX,YB+DY),3
510      NEXT
520      P=M(II,0,1):'center
530      X=X(P):Y=Y(P):Z=Z(P)
535      GOSUB 1000
540      PAINT(X2+DX,Y2+DY),0,3
550      NEXT
600      W$=INPUT$(1):END
1000     '** Rotation subroutine **
1010     CB=COS(B*F):SB=SIN(B*F)
1020     CC=COS(C*F):SC=SIN(C*F)
1030     X1=X*CB+Z*SB
1040     Y1=Y
1050     Z1=-X*SB+Z*CB
1060     X2=X1
1070     Y2=Y1*CC-Z1*SC
1080     Z2=Y1*SC+Z1*CC
1090     RETURN
5000     DATA 24
5010     DATA 0,15,0,10,0,0,20,15,0,20,
           45,0,0,45,0
5020     DATA 0,15,30,10,0,30,20,15,30,
           20,45,30,0,45,30
5030     DATA 20,30,15,30,30,15,10,45,15,
           10,55,15,10,30,30,10,30,40
5040     DATA 5,7,15,-5,0,15,15,7,15,25,
           0,15,10,30,0,10,30,-10,0,
           30,15,-10,30,15
5050     DATA 7
5060     DATA 5,1,2,2,3,3,4,4,5,5,1,21,22
5070     DATA 4,1,6,6,10,10,5,5,1,23,24
5080     DATA 4,1,2,2,7,7,6,6,1,17,18
5090     DATA 4,2,3,3,8,8,7,7,2,19,20
5100     DATA 4,3,4,4,9,9,8,8,3,11,12
5110     DATA 4,4,9,9,10,10,5,5,4,13,14
5120     DATA 5,6,7,7,8,8,9,9,10,10,6,15,16

```

PROGRAMME 11.41

Le moyen de manipuler ce genre d'objet est de diviser les polygones en deux polygones — ou plus — plus petits. La figure 11.91 montre la façon dont l'un des deux polygones a été divisé en deux qui peuvent facilement être traités avec la technique décrite ci-dessus.

Quand un objet consiste en deux parties indépendantes ou plus, cette élimination des lignes et des surfaces cachées ne marchera pas correctement parce que certaines des surfaces des objets qui sont derrière d'autres auront des vecteurs normaux pointant dans le sens Z, tandis que d'autres surfaces situées au-dessus de ces surfaces auront des vecteurs normaux pointant dans le sens -Z. La figure 11.92 montre ce cas : la surface A

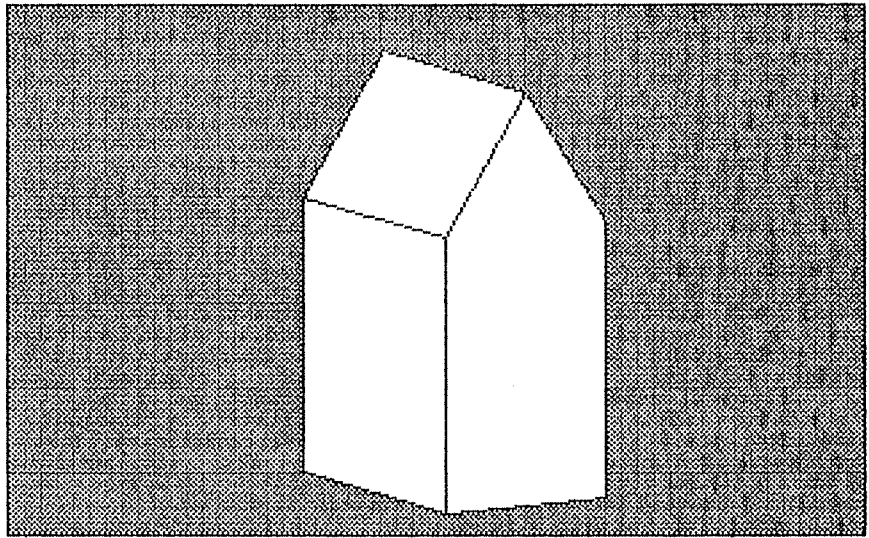


Fig. 11.88

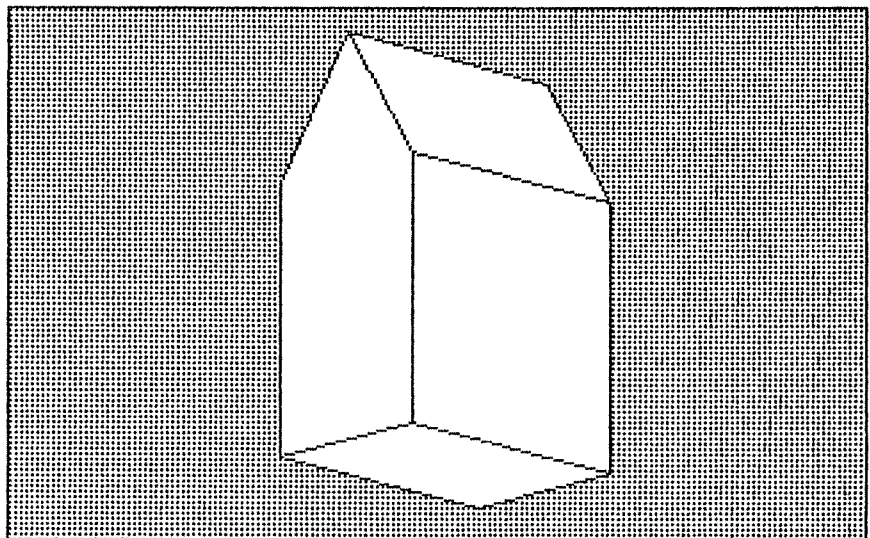


Fig. 11.89

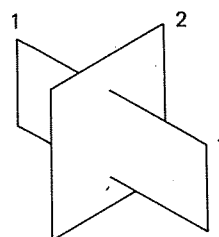


Fig. 11.90

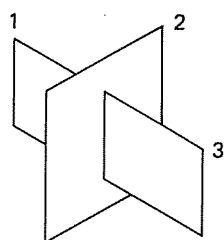


Fig. 11.91

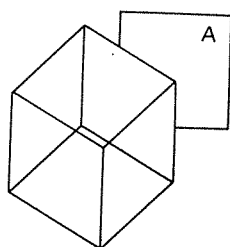


Fig. 11.92

sera dessinée avant la totalité du cube de devant parce que son vecteur normal est parallèle à l'axe des Z, tandis que les autres, bien que certains pointent dans le sens Z, seront triés derrière A.

Cette situation peut être résolue de deux façons, chacune des deux exige une représentation interne différente. Chaque objet sera identifié de telle sorte que le programme puisse savoir quelles surfaces correspondent à quels objets, et un centre (un point situé approximativement au centre de l'objet) est associé à chaque objet.

Dans la première méthode, le tri des polygones est effectué sur chaque objet indépendamment, et les objets sont ensuite triés sur les coordonnées Z de leurs centres. Les objets sont alors tracés selon la méthode utilisée au programme 11.40, en partant de l'objet ayant la plus petite coordonnée Z en son centre et en continuant jusqu'à la plus grande coordonnée Z.

Dans la seconde méthode, outre d'avoir un centre pour chaque objet, on utilise un second point pour définir un vecteur. Comme pour les vecteurs normaux, celui-ci indiquera le sens de l'objet par rapport au centre de tout l'ensemble d'objets. Avec ces vecteurs, l'espace en entier peut être considéré comme un simple objet dont les murs sont à leur tour des objets aussi, chacun desquels aura un vecteur indiquant sur quel côté de « l'univers » il est situé. Le tri des objets est alors exécuté (après les rotations et les transformations de perspective) basé sur l'orientation de ces objets.

Il existe plusieurs méthodes pour effacer les lignes et les surfaces cachées, certaines simples mais inefficaces, d'autres compliquées et plus

précises. On peut trouver deux méthodes intéressantes dans le livre de Roy E. Myers « Microcomputer Graphics » (édité par Addison-Wesley en 1982) et dans « Graphic Software for Microcomputers » de B.J. Korites (chez Duxbury, 1982).

11.2.8 Surfaces transparentes dans les objets solides

Les surfaces des objets solides ne sont pas toutes opaques. Il y a des trous, des fenêtres, des portes ouvertes, etc., des surfaces ou des parties de surfaces qui sont limitées par un polygone mais qui sont transparentes. Si des surfaces sont entièrement transparentes, deux cas peuvent se présenter :

1. Le polygone est complètement entouré par d'autres polygones qui ne sont pas transparents. Dans ce cas, on ne doit rien faire parce que les murs limitants sont dessinés et les lignes qui déterminent le polygone transparent sont également tracées. La figure 11.93 montre une maison avec un mur transparent.

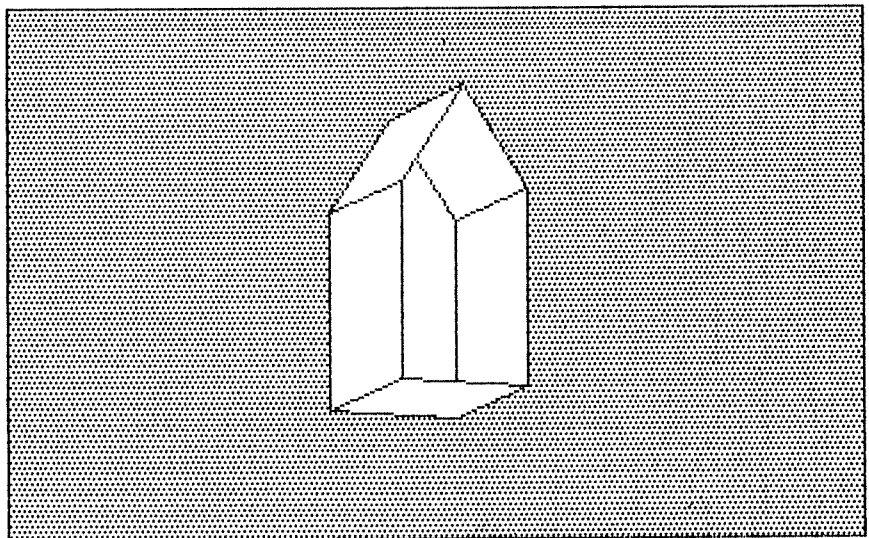


Fig. 11.93

2. Le polygone est transparent et n'est pas limité par d'autres polygones opaques, ou est séparé d'autres polygones. Dans ce cas il est nécessaire d'avoir un tableau supplémentaire pour indiquer la condition de chaque polygone. Si nous disons que 0 signifie opaque et 1 transparent, chaque polygone où l'on trouve un 1 doit être tracé mais non colorié.

Quand seule une partie du polygone est transparente (par exemple une fenêtre dans un mur), il faut utiliser une technique différente. Le polygone avec la partie transparente (la fenêtre) est défini de la même manière qu'un polygone opaque, mais les polygones qui constituent les fenêtres sont définis ensemble avec lui. Le centre doit être défini dans une des parties opaques du polygone. Quand on utilise la méthode bestiale, le polygone externe, de même que tous les polygones qui définissent les fenêtres, sont dessinés et alors le coloriage peut commencer. Le chemin de PAINT est arrêté par les parties internes transparentes. Ainsi le coloriage ne va pas sur les zones à l'intérieur des fenêtres, et celles-ci restent transparentes.

Le programme 11.42 trace une maison avec des murs opaques, des fenêtres transparentes et une porte. Si on utilise les bons angles, il est possible de voir à travers la maison en alignant les fenêtres des murs opposés, comme le montrent les figures 11.94 et 11.95.

```

0  ** 11-42 : Transparent doors and windows **
3  F=1.745329E-02
5  B=150
6  C=180
7  FAC=40
8  DX=230: DY=170
9  VANISHING.POINT=-60: DISTANCE=220
10 SCREEN 1: COLOR 0,0: LINE(0,0)-(319,199),2,BF
20 DIM X(100),Y(100),Z(100),M(50,16,2),
    TF(50),ZT(50)
22 FOR I=1 TO 50:
    TF(I)=I:
    NEXT
30 READ N
40 FOR I=1 TO N
50   READ X(I),Y(I),Z(I): Z(I)=3-Z(I):
    X(I)=X(I)+1.5
55   X(I)=FAC*X(I): Y(I)=FAC*Y(I):
    Z(I)=FAC*Z(I)
60 NEXT
70 READ NP
80 FOR I=1 TO NP
90   READ M(I,0,0)
100  FOR J=1 TO M(I,0,0)
110   READ M(I,J,1),M(I,J,2)
120  NEXT
130  READ M(I,0,1),M(I,0,2)
140 NEXT
150 FOR I=1 TO NP
155   P=M(I,0,1)
156   X=X(P): Y=Y(P): Z=Z(P)
157   GOSUB 1000
160   ZT(I)=Z2
165 NEXT
170 FOR I=1 TO NP-1
171   FOR J=I+1 TO NP
174     P=M(TF(I),0,2): X=X(P): Y=Y(P): Z=Z(P)
175     GOSUB 1000: ZA=Z2-ZT(TF(I))
176     P=M(TF(J),0,2): X=X(P): Y=Y(P): Z=Z(P)

```

```

177     GOSUB 1000:ZB=Z2-ZT(TF(J))
178     IF ZA>ZB
        THEN
            SWAP TF(I),TF(J)
185     NEXT
190 NEXT
300 FOR I=1 TO NP:
    II=TF(I)
310     FOR L=1 TO 1
320         FOR K=1 TO 1
330             FOR J=1 TO M(II,0,0)
340                 P=M(II,J,1)
350                 X=X(P):Y=Y(P):Z=Z(P)
355                 GOSUB 1000:XA=X2:YA=Y2
360                 P=M(II,J,2)
370                 X=X(P):Y=Y(P):Z=Z(P)
375                 GOSUB 1000:XB=X2:YB=Y2
380                 LINE(XA+DX,YA+DY)-(XB+DX,YB+DY),K
390             NEXT
400             P=M(II,0,1):'center
410             X=X(P):Y=Y(P):Z=Z(P)
415             GOSUB 1000
420             PAINT(X2+DX,Y2+DY),K,K
430         NEXT
440     NEXT
450     FOR J=1 TO M(II,0,0)
460         P=M(II,J,1)
470         X=X(P):Y=Y(P):Z=Z(P)
475         GOSUB 1000:XA=X2:YA=Y2
480         P=M(II,J,2)
490         X=X(P):Y=Y(P):Z=Z(P)
495         GOSUB 1000:XB=X2:YB=Y2
500         LINE(XA+DX,YA+DY)-(XB+DX,YB+DY),3
510     NEXT
520     P=M(II,0,1):'center
530     X=X(P):Y=Y(P):Z=Z(P)
535     GOSUB 1000
540     PAINT(X2+DX,Y2+DY),0,3
550 NEXT
600 W#=INPUT$(1):END
1000 '** Rotation subroutine **
1010 CB=COS(B*F):SB=SIN(B*F)
1020 CC=COS(C*F):SC=SIN(C*F)
1025 GOSUB 2000
1030 X1=X*CB+Z*SB
1040 Y1=Y
1050 Z1=-X*SB+Z*CB
1060 X2=X1
1070 Y2=Y1*CC-Z1*SC
1080 Z2=Y1*SC+Z1*CC
1090 RETURN
2000 '** Adjust perspective **
2010 FACTOR=(Z-VANISHING.POINT)/DISTANCE
2020 X=X*FACTOR:Y=Y*FACTOR
2030 RETURN
5000 DATA 40
5010 DATA 0,0,0,1,0,0,1,2.3,0,2.5,2.3,0,
        2.5,0,0,6,0,0,6,3,0,0,3,0
5020 DATA 3.5,2.3,0,5,2.3,0,5,1,0,3.5,1,
        0,3,1.8,0,3,1.8,-1
5030 DATA 0,0,3,0,3,3,6,3,3,6,0,3,.8,1,

```

```

3, .8, 2.3, 3, 2.2, 2.3, 3
5040 DATA 2.2, 1, 3, 3.8, 2.3, 3, 5.2, 2.3, 3,
5.2, 1, 3, 3.8, 1, 3, 3, 1.8, 3, 3, 1.8, 4
5050 DATA 0, 1.8, 1.5, -1, 1.8, 1.5
5060 DATA 6, 1.8, 1.5, 7, 1.8, 1.5, 3, 0, 1.5,
3, -1, 1.5
5070 DATA 3, 4, 0.7, 3, 4.6, 0, 3, 4, 2.3, 3, 4.6, 3
5080 DATA 6, 5.5, 1.5, 0, 5.5, 1.5
5085 DATA 7
5090 DATA 11, 1, 6, 6, 7, 7, 8, 8, 1, 2, 3, 3, 4, 4, 5,
10, 11, 11, 12, 12, 9, 9, 10, 13, 14
5100 DATA 12, 15, 16, 16, 17, 17, 18, 18, 15, 19,
20, 20, 21, 21, 22, 22, 19, 23, 24, 24,
25, 25, 26, 26, 23, 27, 28
5110 DATA 5, 1, 8, 8, 40, 40, 16, 16, 15, 15, 1, 29, 30
5120 DATA 5, 6, 7, 7, 39, 39, 17, 17, 18, 18, 6, 31, 32
5130 DATA 4, 1, 6, 6, 18, 18, 15, 15, 1, 33, 34
5140 DATA 4, 7, 8, 8, 40, 40, 39, 39, 7, 35, 36
5150 DATA 4, 17, 39, 39, 40, 40, 16, 16, 17, 37, 38

```

PROGRAMME 11.42

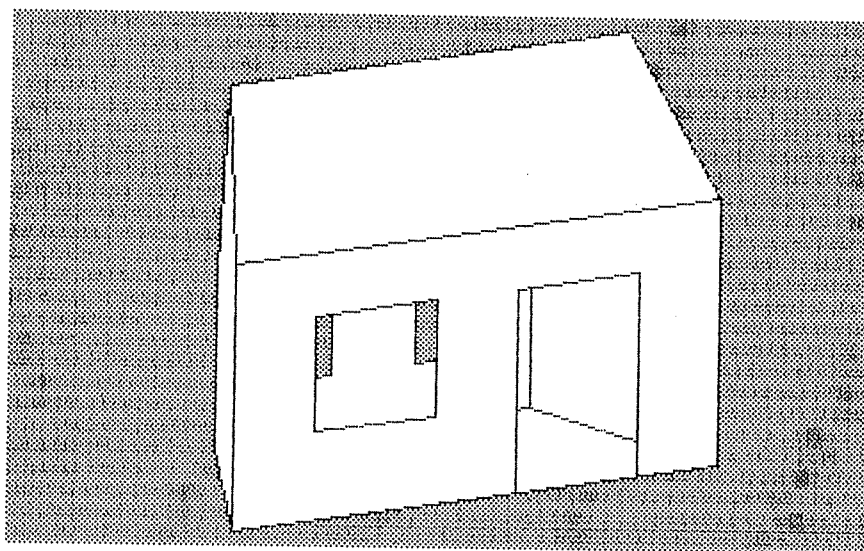


Fig. 11.94

Quand le polygone est transparent avec plusieurs fenêtres opaques à l'intérieur, le grand polygone doit être traité comme s'il était complètement transparent et les plus petits doivent être définis comme des polygones séparés opaques.

La méthode pour enlever les lignes et les surfaces cachées qui est présentée ici est une des plus simples et des plus faciles à comprendre. Toutefois, elle ne peut pas s'appliquer à certains cas spéciaux, ni aux objets définis par des algorithmes et non par des points individuels qui peuvent être relevés facilement pour former des polygones.

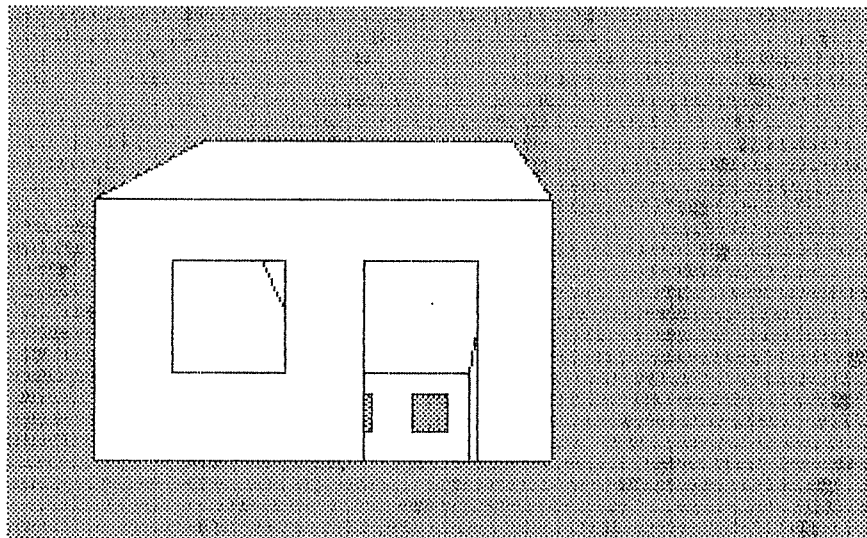


Fig. 11.95

11.2.9 Solides circulaires

Dans ce paragraphe nous décrirons toute une famille d'objets tridimensionnels qui sont mieux décrits par des algorithmes que par des lignes ou des polygones.

Le premier exemple, le programme 11.43, dessine une spirale en calculant les valeurs X et Y avec les coordonnées polaires et un rayon qui croît progressivement, comme cela a été fait dans le programme 3.3.1. Si on prend le rayon comme coordonnée Z de chaque point, la spirale sera élevée d'une manière croissante à partir du plan $X-Y$. La figure 11.39 montre la spirale produite avec deux angles de 45 degrés.

```

0  '** 11-43 : Conic spiral **
5  F=1.745329E-02
10 SCREEN 1:CLS
12 INPUT"Rotation angles (around X,Y) ";B,C:
   CLS
20 FOR I=0 TO 85 STEP .1
30   R=I*1.1:Z=R
40   X=R*COS(I)
50   Y=R*SIN(I):GOSUB 1000
60   IF I=0
       THEN
           PSET(80+X2,130+Y2)
       ELSE
           LINE-(80+X2,130+Y2)
70 NEXT
80 END
1000 '** Rotation subroutine **

```



```

1010 CB=cos(B*F):SB=sin(B*F)
1020 CC=cos(C*F):SC=sin(C*F)
1030 X1=X*CB+Z*SB
1040 Y1=Y
1050 Z1=-X*SB+Z*CB
1060 X2=X1
1070 Y2=Y1*CC-Z1*SC
1080 Z2=Y1*SC+Z1*CC
1090 RETURN

```

PROGRAMME 11.43

Le second exemple dessine une fleur : en utilisant une fonction polaire, le programme 11.44 trace une fleur à quatre pétales. Le rayon est également la hauteur de chaque point sur l'axe des Z ; ainsi quand les points se rapprochent du centre, la fleur est profonde (Z étant proche de 0) et les pétales sont élevés au-dessus du plan X-Y (Z étant proche du rayon maximum). Comme le rayon devient négatif dans ce programme, on a utilisé la valeur absolue pour la coordonnée Z.

```

0  ** 11-44 : Flower **
10 F=1.745329E-02
20 SCREEN 1:COLOR 1,0:CLS
30 INPUT"Rotation angles (around X,Y) ";B,C
40 LINE(0,0)-(319,199),2,BF
45 FOR I=0 TO 6.3 STEP .05
50   R=90*SIN(2*I):Z=ABS(R)
60   X=R*cos(I)
70   Y=R*sin(I)
80   GOSUB 1000
90   IF I>0
       THEN
           LINE(PX,PY)-(X2,Y2)
100  PX=X2:PY=Y2
110 NEXT I:PAINT(0,0),0,3
120 W$:INPUT$(1):END
1000 ** Rotation subroutine **
1010 CB=cos(B*F):SB=sin(B*F)
1020 CC=cos(C*F):SC=sin(C*F)
1030 X1=X*CB+Z*SB
1040 Y1=Y
1050 Z1=-X*SB+Z*CB
1060 X2=X1
1070 Y2=Y1*CC-Z1*SC
1080 Z2=Y1*SC+Z1*CC
1090 X2=XD+X2+160:Y2=YD+100-Y2
1100 RETURN

```

PROGRAMME 11.44

La figure 11.96 montre la fleur sans rotations.

Les figures 11.97 et 11.98 montrent deux vues différentes de la fleur.

Dans notre troisième exemple de solides circulaires, nous présenterons une technique permettant la production de plusieurs formes remarquables. On peut créer facilement des formes très proches d'objets ayant une symé-

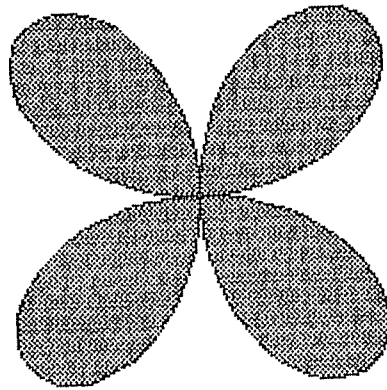


Fig. 11.96

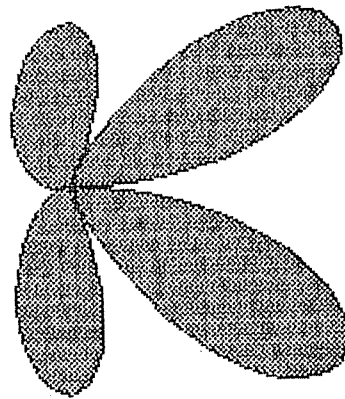


Fig. 11.97

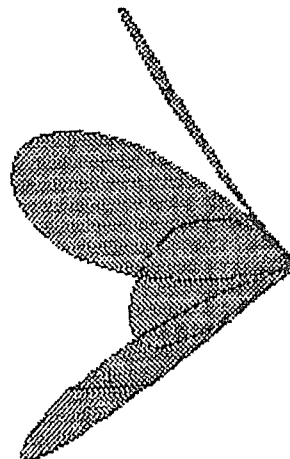


Fig. 11.98

trie circulaire avec des anneaux polygonaux interconnectés. Ne vous en faites pas si le nom fait peur : c'est vraiment très simple et encore plus facile à faire. On peut voir sur la figure 11.99 qu'un cylindre peut être représenté par deux cercles reliés.

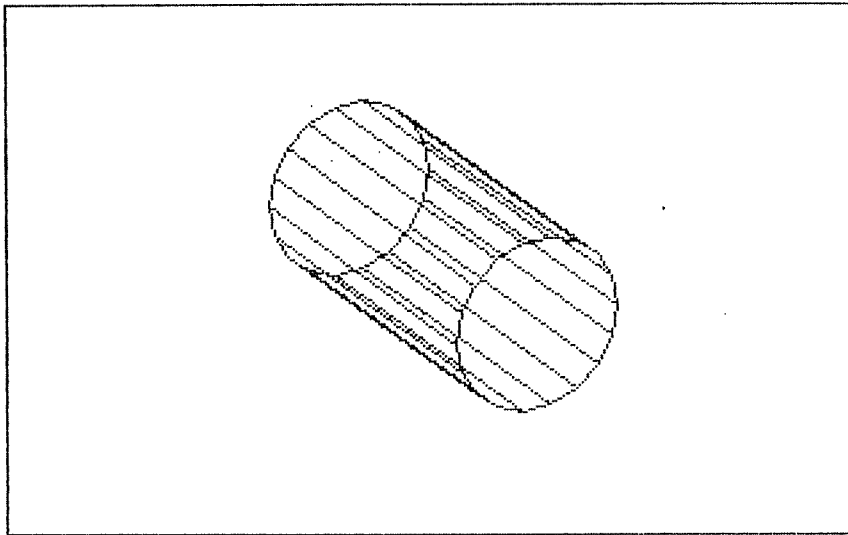


Fig. 11.99

Nous allons tracer le cylindre en calculant les points d'un cercle à l'aide des coordonnées polaires. A chaque valeur X et Y nous ajouterons un Z positif pour tracer un cercle au-dessus du plan X-Y, et un Z négatif pour tracer un cercle en dessous du plan X-Y. Si en plus de relier les points successifs d'un cercle, nous relierons ensemble les points ayant les mêmes valeurs X et Y, nous aurons l'approximation d'un cylindre plutôt bonne, comme celle créée par le programme 11.45 et que l'on voit à la figure 11.99.

```
0 ' ** 11-45 : Cylinder **
10 F=1.745329E-02
30 SCREEN 1:CLS:DIM X(20),Y(20)
40 B=-30:C=-25:READ N
50 FOR I=1 TO N
60   READ R,Z:S=1
70   FOR A=0 TO 360 STEP 20:ANG=A*F
80     X=R*COS(ANG):Y=R*SIN(ANG)
90     GOSUB 1000
100    IF I>1
        THEN
          LINE(X(S),Y(S))-(X2,Y2)
110    IF A>0
        THEN
          LINE(PX,PY)-(X2,Y2)
```

```

120   PX=X2:PY=Y2:X(S)=X2:Y(S)=Y2
130   S=S+1
150 NEXT
160 NEXT
170 W$=INPUT$(1):END
1000 '** Rotation subroutine **
1010 CB=COS(B*F):SB=SIN(B*F)
1020 CC=COS(C*F):SC=SIN(C*F)
1030 X1=X*CB+Z*SB
1040 Y1=Y
1050 Z1=-X*SB+Z*CB
1060 X2=X1
1070 Y2=Y1*CC-Z1*SC
1080 Z2=Y1*SC+Z1*CC
1090 X2=XD+X2+160:Y2=YD+100-Y2
1100 RETURN
5000 DATA 2
5010 DATA 35,0
5020 DATA 35,140

```

PROGRAMME 11.45

La forme suivante que nous allons réaliser grâce à cette méthode est un diamant du type de celui que l'on voit sur la partie gauche de la figure 11.100.

La forme de droite est une vue de côté du diamant. Notez que tout l'objet est fait d'anneaux polygonaux dont nous avons besoin de connaître seulement la hauteur et le rayon. Le programme 11.46 lit les trois rayons et hauteurs à la ligne 60. En utilisant la technique du programme 11.45, une boucle FOR trace un polygone (un décagone) à la hauteur Z et mémorise les coordonnées X et Y de chaque point. Quand le polygone suivant est tracé, les points sont non seulement reliés entre eux, mais aussi avec ceux de l'anneau précédent. La figure 11.101 montre le diamant après une rotation autour de l'axe des X de -5 degrés et autour de l'axe Y de -95 degrés.

```

0 '** 11-46 : Diamond **
10 F=1.745329E-02:B=-5:C=-95
20 CLS:INPUT"factor ";FAC
30 SCREEN 1:CLS:DIM X(11),Y(11)
40 CLS:RESTORE:READ N
50 FOR I=1 TO N
60   READ R,Z:R=R*FAC:Z=Z*FAC:S=1
70   FOR A=0 TO 360 STEP 36:ANG=A*F
80     X=R*COS(ANG):Y=R*SIN(ANG)
90     GOSUB 1000
100    IF I>1
110      THEN
120        LINE(X(S),Y(S)+60)-(X2,Y2+60)
130      IF A>0
140        THEN
150          LINE(PX,PY+60)-(X2,Y2+60)
160        PX=X2:PY=Y2:X(S)=X2:Y(S)=Y2
170        S=S+1
180      NEXT A
190    NEXT I
200  NEXT

```

```

160 NEXT
170 W$=INPUT$(1):END
1000 '** Rotation subroutine **
1010 CB=cos(B*F):SB=sin(B*F)
1020 CC=cos(C*F):SC=sin(C*F)
1030 X1=X*CB+Z*SB
1040 Y1=Y
1050 Z1=-X*SB+Z*CB
1060 X2=X1
1070 Y2=Y1*CC-Z1*SC
1080 Z2=Y1*SC+Z1*CC
1090 X2=XD+X2+160:Y2=YD+100-Y2
1100 RETURN
5000 DATA 3
5010 DATA 30,0,45,20,30,40

```

PROGRAMME 11.46

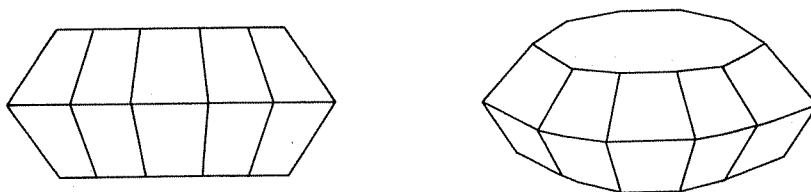


Fig. 11.100

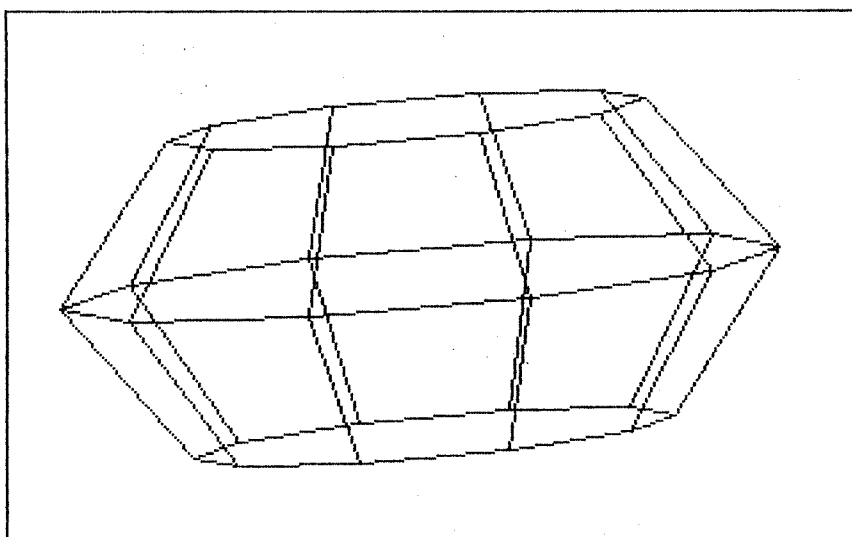


Fig. 11.101

La figure 11.102 montre une vue de côté d'un calice. Si nous prenons les rayons et hauteurs approximatifs (que l'on voit à la table 11.5) et les faisons rentrer comme données dans le programme 11.46, celui-ci reconstruira le calice.

Table 11.5

ANNEAU	RAYON	Z	ANNEAU	RAYON	Z
1	0	-6	9	5.82	1.45
2	1.35	-5.85	10	5.35	2.72
3	2.63	-5.39	11	4.6	3.86
4	3.78	-4.66	12	3.61	4.79
5	4.73	-3.69	13	2.44	5.48
6	5.44	-2.54	14	1.14	5.89
7	5.87	-1.25	15	0	6
8	6	0.1			

Le programme 11.47 est une version améliorée du programme 11.46. Outre un facteur d'échelle qui permet des changements de taille de l'objet, on peut spécifier un déplacement X et Y pour centrer le graphique. En appuyant sur <ESC>, vous pouvez provoquer l'apparition d'un petit menu, indiquant les valeurs du facteur en cours et des déplacements X et Y et permettant de changer ces valeurs.

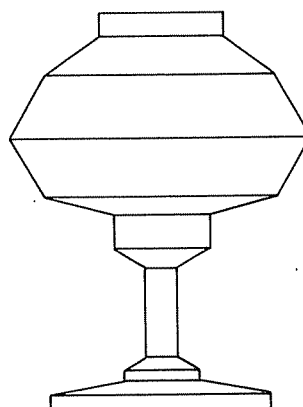


Fig. 11.102

```
0  '*** 11-47 : Circular solid **
10 F=1.745329E-02
20 CLS:INPUT"factor ";FAC
30 SCREEN 1:CLS:DIM X(11),Y(11)
40 CLS:RESTORE:READ N
50 FOR I=1 TO N
60   READ R,Z:R=R*FAC:Z=Z*FAC:S=1
```

```

70   FOR A=0 TO 360 STEP 36:ANG=A*F
80   X=R*COS(ANG):Y=R*SIN(ANG)
90   GOSUB 1000
100  IF I>1
    THEN
      LINE(X(S),Y(S))-(X2,Y2)
110  IF A>0
    THEN
      LINE(PX,PY)-(X2,Y2)
120  PX=X2:PY=Y2:X(S)=X2:Y(S)=Y2
130  S=S+1
140  W$=INKEY$:
    IF W$<>" "
    THEN
      510
150  NEXT
160  NEXT
500  W$=INPUT$(1)
510  V=VAL(W$)
520  IF W$<>CHR$(27)
    THEN
      590
    ELSE
      CLS:PRINT"1-X displacement ";XD:
      PRINT"2-Y displacement "YD
530  PRINT"3-Factor "FAC:PRINT"4-Continue ";
540  W$=INPUT$(1):
    IF W$<"1"OR W$>"4"
    THEN
      540
    ELSE
      ON VAL(W$)GOTO 550,560,570,580
550  INPUT XD:GOTO 40
560  INPUT YD:GOTO 40
570  INPUT FAC:GOTO 40
580  GOTO 40
590  IF V=8
    THEN
      C=C-5:CLS:GOTO 40
600  IF V=2
    THEN
      C=C+5:CLS:GOTO 40
610  IF V=4
    THEN
      B=B-5:CLS:GOTO 40
620  IF V=6
    THEN
      B=B+5:CLS:GOTO 40
630  BEEP:GOTO 500
1000  '** Rotation subroutine **
1010  CB=COS(B*F):SB=SIN(B*F)
1020  CC=COS(C*F):SC=SIN(C*F)
1030  X1=X*CB+Z*SB
1040  Y1=Y
1050  Z1=-X*SB+Z*CB
1060  X2=X1
1070  Y2=Y1*CC-Z1*SC
1080  Z2=Y1*SC+Z1*CC
1090  X2=XD+X2+160:Y2=YD+100-Y2
1100  RETURN
5000  DATA 14

```

```

5010 DATA 2.2,0,2.2,1,4,2,4.7,4,3.5,6
5020 DATA 1.5,6.5,1.5,7.5,1,7.75,.5,8
5030 DATA .5,11,1.2,11.2,1.2,11.5
5040 DATA 3.5,12,3.5,12.5

```

PROGRAMME 11.47

Les figures 11.103, 11.104 et 11.105 montrent le calice avec trois orientations différentes.

Notre dernier exemple de formes circulaires sera une sphère. Au lieu de cercles, cet objet sera fait de polygones, aussi ce ne sera pas vraiment une sphère, de même que les cercles tracés en coordonnées polaires ne sont pas vraiment des cercles. La table 11.6 montre les valeurs utilisées pour les rayons et les hauteurs. Elles ont été calculées de la même manière que nous avons calculé X et Y pour tracer un cercle avec les coordonnées polaires. La figure 11.106 montre le graphique résultant.

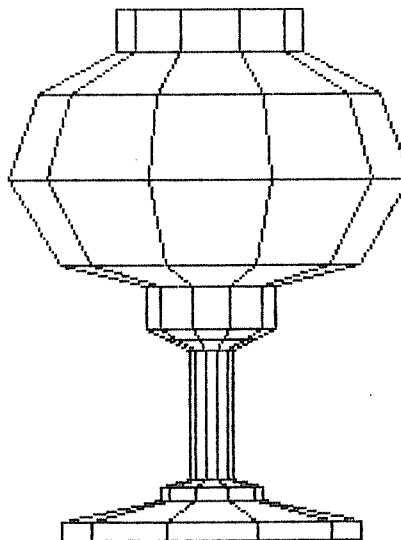


Fig. 11.103

Table 11.6

ANNEAU	RAYON	Z	ANNEAU	RAYON	Z
1	2.2	0	8	1	7.75
2	2.2	1	9	0.5	8
3	4	2	10	0.5	11
4	4.7	4	11	1.2	11.2
5	3.5	6	12	1.2	11.5
6	1.5	6.5	13	3.5	12
7	1.5	7.5	14	3.5	12.5

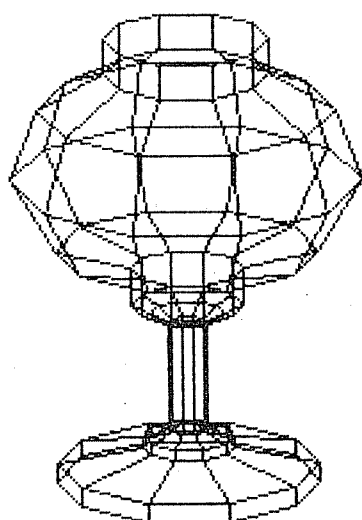


Fig. 11.104

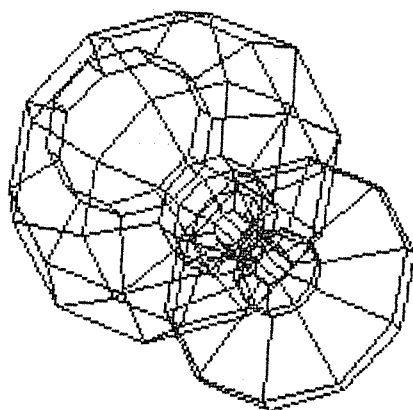


Fig. 11.105

11.2.10 Animation d'objets 3-D

Nous présentons ici un programme dans lequel un objet tridimensionnel est animé. La seule manière d'animer avec succès un objet en BASIC est à l'aide des instructions GET et PUT. Les calculs de rotations et de perspective sont trop longs pour permettre une création et un affichage immédiats. Au programme 11.48 nous utilisons le calice produit par le programme 11.47. On utilise dix-huit tableaux, de MA à MR pour mémoriser les différentes formes. La production des figures prend du temps,

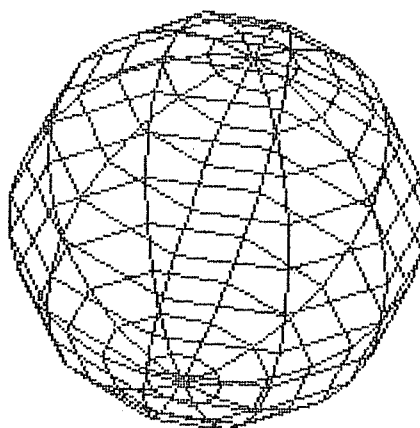


Fig. 11.106

mais une fois qu'elles sont dans les tableaux, la boucle FOR des lignes 420 à 780 les écrit (PUT) successivement et rapidement sur l'écran (sous le mode PSET) pour créer une animation très réaliste. Notez que les tableaux sont affichés dans un ordre spécial : la boucle commence avec MA, MB, etc. mais après MR, l'ordre est renversé pour aller de MQ, MP, etc. et revenir à MB où la boucle redémarre. Le calice a été tourné de 180 degrés, ou un demi-tour autour de son centre (situé quelque part dans le pied). Si les tableaux sont affichés de MA à MR et qu'on répète le processus, il y aura un saut. Si, d'autre part, l'ordre est renversé, le calice basculera en arrière et le mouvement sera très lent.

```

0  ** 11-48 : 3-D animation **
10 F=1.745329E-02:G=1:C=90:INB=9.473599
20 DIM MA(275),MB(275),MC(275),MD(275),
    ME(275),MF(275),MG(275),MH(275),
    MI(275),MJ(275),MK(275),ML(275),
    MM(275),MN(275),MO(275),MP(275),
    MQ(275),MR(275)
30 CLS:FAC=5
40 SCREEN 1:CLS:DIM X(11),Y(11)
50 FOR C=-90 TO 80 STEP INB
60   CLS:RESTORE:READ N
70   FOR I=1 TO N
80     READ R,Z:R=R*FAC:Z=Z-6.25:
        Z=Z*FAC:S=1
90     FOR A=0 TO 360 STEP 36:
        ANG=A*F
100      X=R*COS(ANG):Y=R*SIN(ANG)
110      GOSUB 1000
120      IF (Z>-22 AND Z<6) OR Z>27
          THEN
            COL=2
          ELSE
            COL=1

```

```

130     IF I>1
        THEN
            LINE(X(S),Y(S))-(X2,Y2),COL
140     IF A>0
        THEN
            LINE(PX,PY)-(X2,Y2),COL
150     PX=X2:PY=Y2:X(S)=X2:Y(S)=Y2
160     S=S+1
170     NEXT
180     NEXT
190     ON G GOTO
        200,210,220,230,240,250,260,
        270,280,290,300,310,320,330,
        340,350,360,370
200     GET(136,64)-(186,140),MA:GOTO 390
210     GET(136,64)-(186,140),MB:GOTO 390
220     GET(136,64)-(186,140),MC:GOTO 390
230     GET(136,64)-(186,140),MD:GOTO 390
240     GET(136,64)-(186,140),ME:GOTO 390
250     GET(136,64)-(186,140),MF:GOTO 390
260     GET(136,64)-(186,140),MG:GOTO 390
270     GET(136,64)-(186,140),MH:GOTO 390
280     GET(136,64)-(186,140),MI:GOTO 390
290     GET(136,64)-(186,140),MJ:GOTO 390
300     GET(136,64)-(186,140),MK:GOTO 390
310     GET(136,64)-(186,140),ML:GOTO 390
320     GET(136,64)-(186,140),MM:GOTO 390
330     GET(136,64)-(186,140),MN:GOTO 390
340     GET(136,64)-(186,140),MO:GOTO 390
350     GET(136,64)-(186,140),MP:GOTO 390
360     GET(136,64)-(186,140),MQ:GOTO 390
370     GET(136,64)-(186,140),MR:GOTO 390
390     G=G+1
400     NEXT
410     CLS
420     FOR I=1 TO 34
430     ON I GOTO
        440,450,460,470,480,490,500,
        510,520,530,540,550,560,570,
        580,590,600,610,620,630,640,
        650,660,670,680,690,700,710,
        720,730,740,750,760,770
440     PUT(50,50),MA,PSET:GOTO 780
450     PUT(50,50),MB,PSET:GOTO 780
460     PUT(50,50),MC,PSET:GOTO 780
470     PUT(50,50),MD,PSET:GOTO 780
480     PUT(50,50),ME,PSET:GOTO 780
490     PUT(50,50),MF,PSET:GOTO 780
500     PUT(50,50),MG,PSET:GOTO 780
510     PUT(50,50),MH,PSET:GOTO 780
520     PUT(50,50),MI,PSET:GOTO 780
530     PUT(50,50),MJ,PSET:GOTO 780
540     PUT(50,50),MK,PSET:GOTO 780
550     PUT(50,50),ML,PSET:GOTO 780
560     PUT(50,50),MM,PSET:GOTO 780
570     PUT(50,50),MN,PSET:GOTO 780
580     PUT(50,50),MO,PSET:GOTO 780
590     PUT(50,50),MP,PSET:GOTO 780
600     PUT(50,50),MQ,PSET:GOTO 780
610     PUT(50,50),MR,PSET:GOTO 780
620     PUT(50,50),MQ,PSET:GOTO 780

```

```

630 PUT (50,50),MP,PSET:GOTO 780
640 PUT (50,50),MO,PSET:GOTO 780
650 PUT (50,50),MN,PSET:GOTO 780
660 PUT (50,50),MM,PSET:GOTO 780
670 PUT (50,50),ML,PSET:GOTO 780
680 PUT (50,50),MK,PSET:GOTO 780
690 PUT (50,50),MJ,PSET:GOTO 780
700 PUT (50,50),MI,PSET:GOTO 780
710 PUT (50,50),MH,PSET:GOTO 780
720 PUT (50,50),MG,PSET:GOTO 780
730 PUT (50,50),MF,PSET:GOTO 780
740 PUT (50,50),ME,PSET:GOTO 780
750 PUT (50,50),MD,PSET:GOTO 780
760 PUT (50,50),MC,PSET:GOTO 780
770 PUT (50,50),MB,PSET
780 NEXT:GOTO 420
1000 '** Rotation subroutine **
1010 CB=COS(B*F):SB=SIN(B*F)
1020 CC=COS(C*F):SC=SIN(C*F)
1030 X1=X*CB+Z*SB
1040 Y1=Y
1050 Z1=-X*SB+Z*CB
1060 X2=X1
1070 Y2=Y1*CC-Z1*SC
1080 Z2=Y1*SC+Z1*CC
1090 X2=XD+X2+160:Y2=YD+100-Y2
1100 RETURN
5000 DATA 14
5010 DATA 2.2,0,2.2,1,4,2,4.7,4,3.5,6
5020 DATA 1.5,6.5,1.5,7.5,1,7.75,.5,8
5030 DATA .5,11,1.2,11.2,1.2,11.5
5040 DATA 3.5,12,3.5,12.5

```

PROGRAMME 11.48

11.2.11 Graphiques stéréo

L'élimination des lignes et des surfaces cachées et la perspective ajoutent du réalisme à la représentation d'objets tridimensionnels. Il existe une méthode pour franchir une nouvelle étape dans cette représentation, basée sur la technique utilisée par les yeux et le cerveau pour détecter la profondeur. Cette technique s'appelle la stéréoscopie. Etant donné que les deux yeux sont écartés, l'image perçue par l'œil gauche est différente de celle perçue par l'œil droit. Le cerveau utilise cette information pour produire une image où la profondeur n'est pas seulement estimée par la perspective ou les ombres, mais est en fait reconstruite très précisément et pratiquement sentie. Si deux images qui ont des différences de perspective sont chacune envoyées d'une certaine façon à un œil différent, le cerveau reconstruira la profondeur. Des instruments spéciaux appelés stéréoscopes sont nécessaires⁷ pour visionner une image différente avec chaque œil. Nous allons décrire deux stéréoscopes.

L'utilisation de stéréoscopes à la maison était très populaire au début du siècle. Aujourd'hui, il y a quelques tentatives sporadiques pour ramener cette popularité avec des films qui malheureusement prêtent beaucoup plus d'attention à l'astuce de l'effet qu'à son véritable contenu. Il y a même des projets pour introduire les stéréoscopes à la télévision.

Pour produire ce type de graphiques avec un ordinateur, deux images doivent être produites avec une légère variation dans l'angle de vue, chaque image correspondant à la vue d'un œil. Quand ces images sont fusionnées avec un dispositif spécial qui sera expliqué plus tard, on verra une image vraiment tridimensionnelle.

Pour la représentation des fonctions, il suffit de tracer la surface deux fois, en augmentant le facteur de décalage horizontal dans une des deux versions. Le programme 11.49 trace en haute résolution la vue en perspective des deux fonctions sinus du programme 11.24. La fonction dessinée sur la partie droite de l'écran est exactement celle de la figure 11.33. Sur le côté gauche, au lieu d'ajouter $X*0,5$ à chaque Y , on prend $X*0,7$ pour produire une perspective légèrement différente. La ligne 90 du programme 11.24 a été changée en

90 $NZ=Z+X*.7+100$

La figure 11.107 montre les deux versions. Bien qu'à première vue elles aient l'air identiques, il existe des petites différences de perspective qui aideront à construire le volume lorsqu'elles seront vues au travers de l'appareil approprié.

```

0  ** 11-49 : Stereoscopic sine **
10 SCREEN 2:CLS
20 MAX.X=100:VANISH.PT=-180:CLS
30 DIST=MAX.X-VANISH.PT
40 FOR X=-100 TO 100 STEP 7
50   FACTOR=(X-VANISH.PT)/DIST
60   FOR Y=-150 TO 150
70     GOSUB 1000
80     NZ=Z+X*.5+100
90     NY1=CINT(Y*FACTOR+X*.2+160):
       IF NY1=PY1
         THEN
           130
100    LINE(NY1,NZ)-(NY1,199),0
110    IF Y>-150
       THEN
         LINE(PY1,PZ1)-(NY1,NZ)
120    PY1=NY1:PZ1=NZ
130    NY2=CINT(Y*FACTOR+480):
       IF NY2=PY2
         THEN
           170
140    LINE(NY2,NZ)-(NY2,199),0
150    IF Y>-150
       THEN
         LINE(PY2,PZ2)-(NY2,NZ)

```

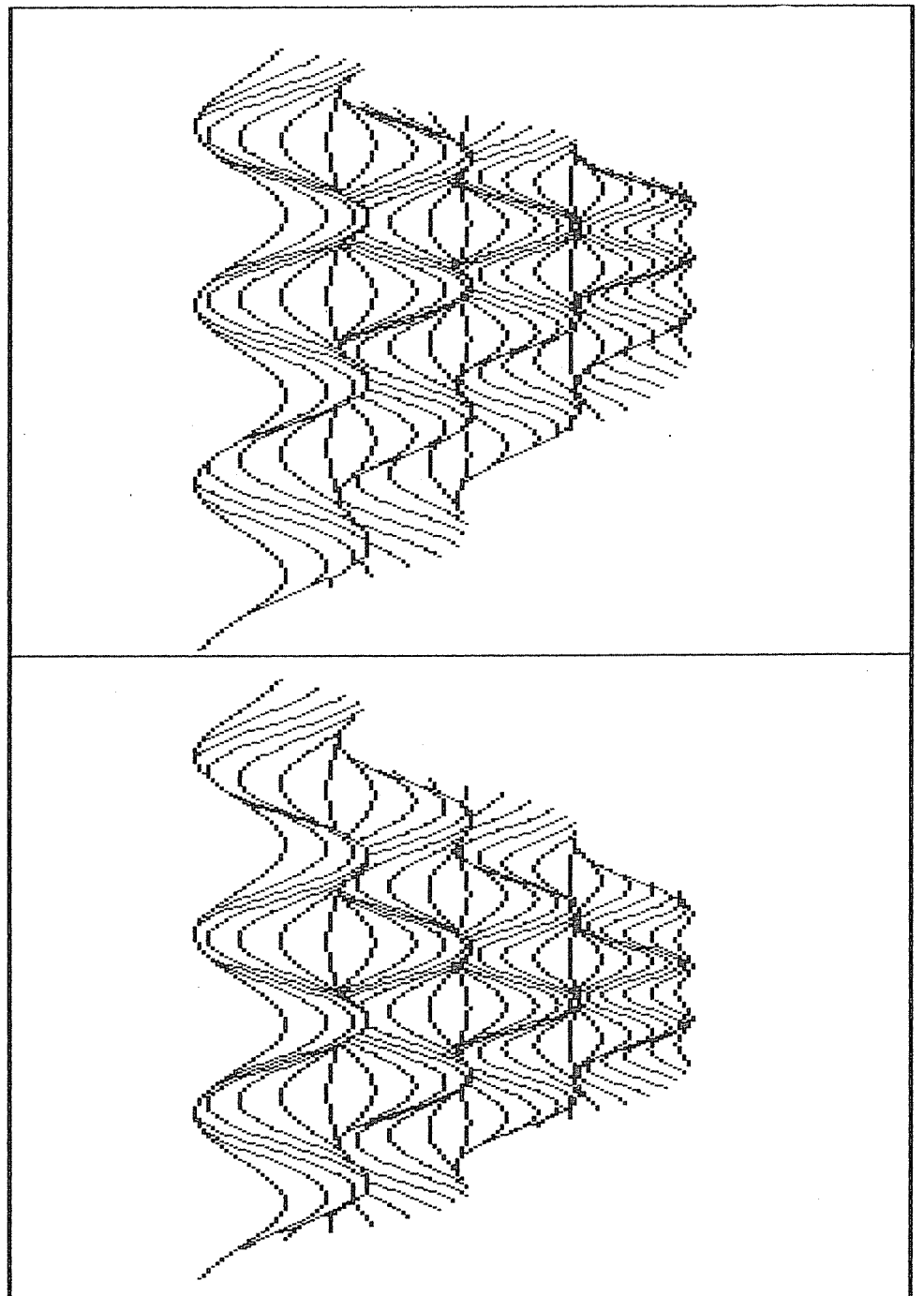


Figure 11.107

```

160     PY2=NY2:PZ2=NZ
170     NEXT
180 NEXT
190 LINE(0,0)-(639,199),3,B:
    LINE(320,0)-(320,199)
200 W$=INPUT$(1)
210 END
1000 'Function
1010 Z=20*SIN(X*.06)*SIN(Y*.07)*FACTOR
1020 RETURN

```

PROGRAMME 11.49

Nous allons maintenant montrer la structure de deux dispositifs pour « voir » ces images. Le premier est le stéréoscope à quatre miroirs⁸ de la figure 11.108. Le cheminement de la ligne de vision de chaque œil est infléchi au moyen de deux miroirs. Si les images stéréoscopiques sont placées chacune en face d'un miroir, elles seront fusionnées en une image vraiment tridimensionnelle. Il est important que les miroirs soient placés à 45 degrés par rapport au fond et au devant de la boîte et soient parfaitement verticaux. On peut utiliser ce dispositif pour visionner la figure 11.107.

L'autre, le stéréoscope à un miroir, ne demande pas d'assemblage spécial, mais les images visionnées avec lui demandent une petite manipulation supplémentaire. Si une des deux images est inversée dans le sens des X, quand on la regardera avec un simple miroir, on la remettra droite. La figure 11.109 montre comment on utilise ce fait pour fusionner les deux images ensemble.

En changeant simplement la ligne 130 du programme 11.49 en

```

130 NY2=CINT(480-Y*FACTOR):
    IF NY2=PY2
    THEN
        170

```

la partie gauche du graphique sera renversée, produisant le graphique vu à la figure 11.110.

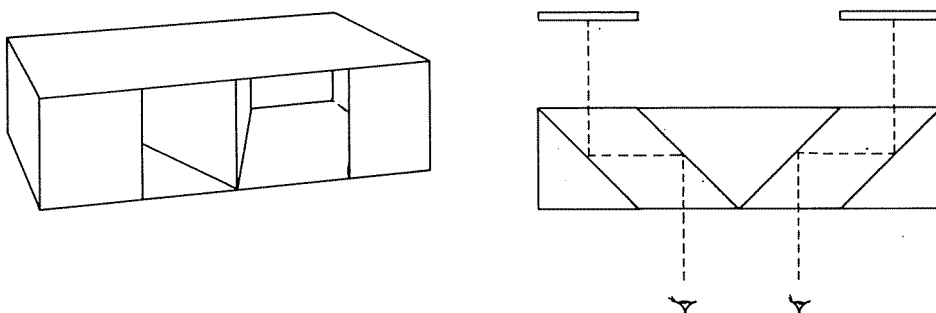


Fig. 11.108

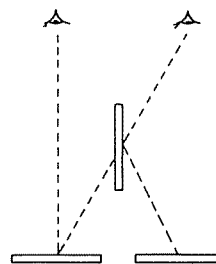


Fig. 11.109

Ici la position de chaque graphique est importante : si l'image qui correspond à l'œil gauche est visionnée avec l'œil gauche, et vice versa, l'illusion correcte de volume sera créée. Si l'ordre est renversé, la profondeur est inversée et l'image a l'air confuse. Dans le stéréoscope à deux miroirs seul l'ordre des images peut être renversé. Dans le stéréoscope à un miroir, non seulement les images, mais aussi le miroir peuvent être invertis. Si la surface réfléchissante est appliquée à l'œil droit, l'effet sera le contraire de ce qu'il serait appliqué à l'œil gauche. L'ordre que nous utilisons suppose que la surface réfléchissante est appliquée à l'œil droit.

Les figures 11.111, 11.112, 11.113 et 11.114 montrent les images stéréoscopiques produites par des programmes déjà étudiés. Nous donnons à la fois la version normale (pour visionner avec un stéréoscope normal à quatre miroirs) et la version inversée. C'est une bonne idée que de faire une photocopie des figures pour faciliter le placement des images individuelles en face de chaque miroir.

Pour produire des graphiques stéréoscopiques d'objets solides, il est nécessaire de générer les deux figures avec une légère différence dans la rotation autour de l'axe des Y, et les calculs de cette rotation doivent être effectués avant ceux de l'axe des X. La figure 11.115 montre le calice du programme 11.47 : une figure a été produite et tournée de 0 degré autour de l'axe des Y, l'autre de 5 degrés. A cause de la symétrie et de la transparence de cette forme, on peut utiliser le même ensemble d'images à la fois pour le stéréoscope à deux miroirs et pour celui à un. On peut aussi les visionner correctement dans n'importe quel ordre.

Les figures 11.116 et 11.117 montrent la fleur du programme 11.44 ayant tourné de 10 degrés autour de l'axe des X, et de 25 et 35 degrés autour de l'axe des Y. Finalement les figures 11.118 et 11.119 montrent les lettres du programme 11.39 étirées verticalement. Ici la différence d'orientation a été réalisée en retranchant 10 de la coordonnée X de chaque point, pour produire une variation dans le point de vue comme celle décrite dans les figures 11.80, 11.81 et 11.82.

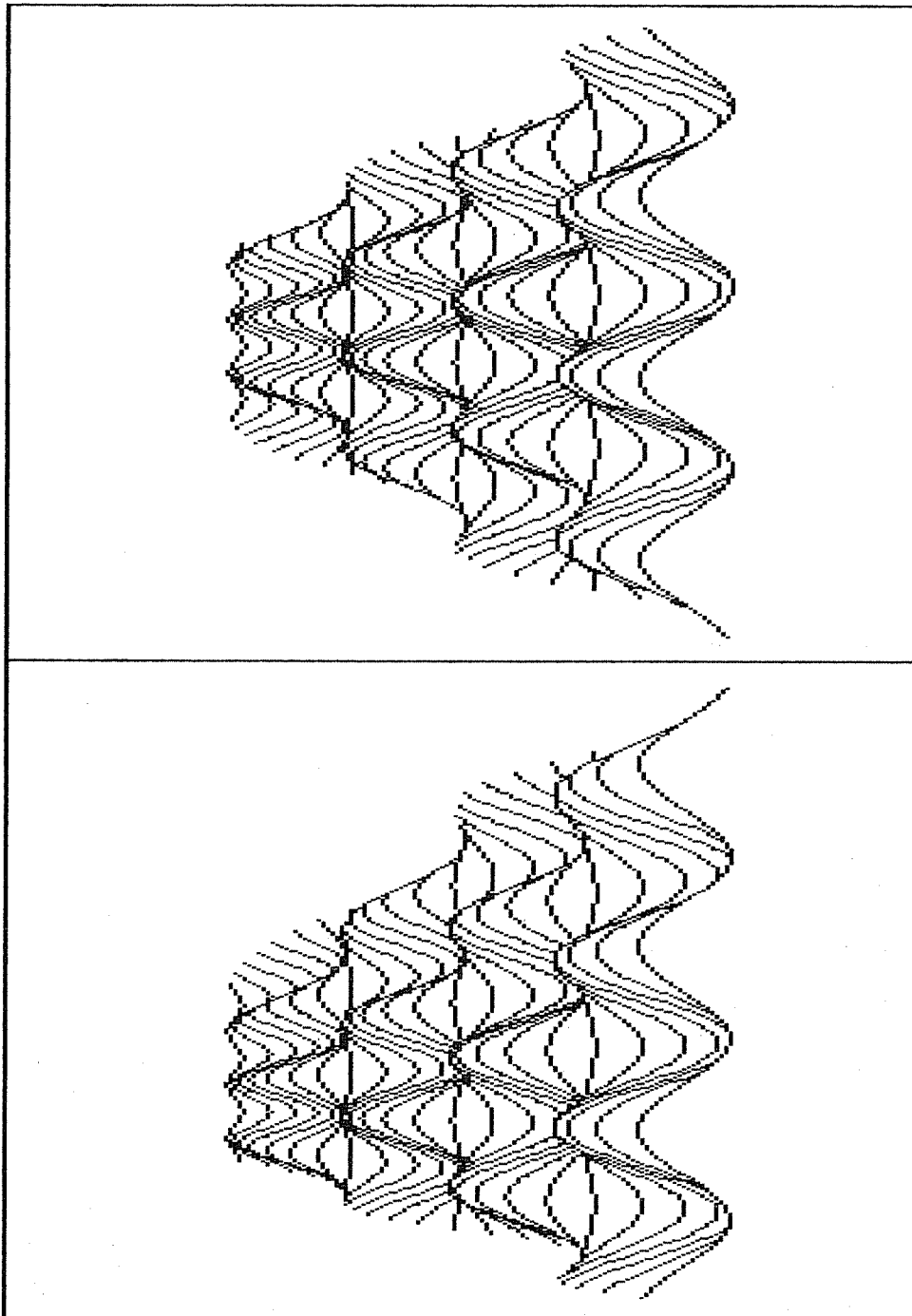


Fig. 11.110

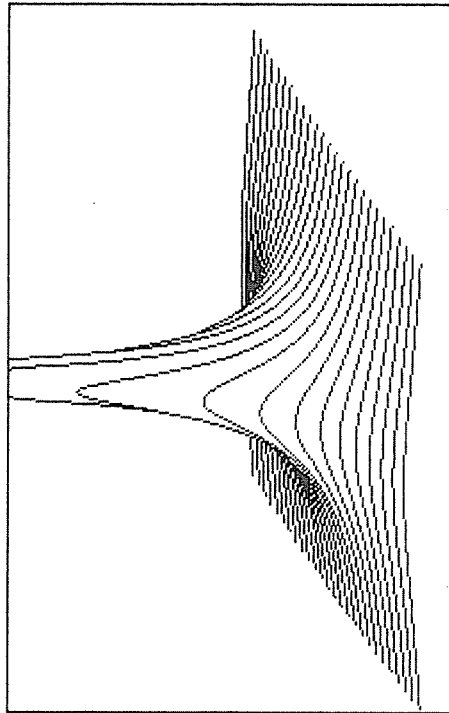


Fig. 11.111

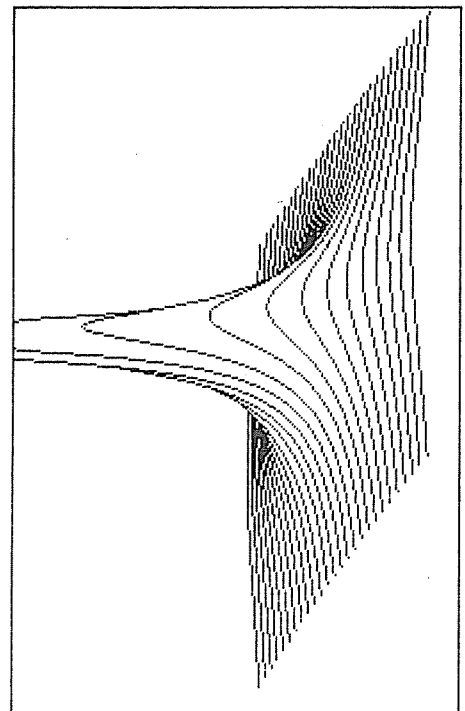
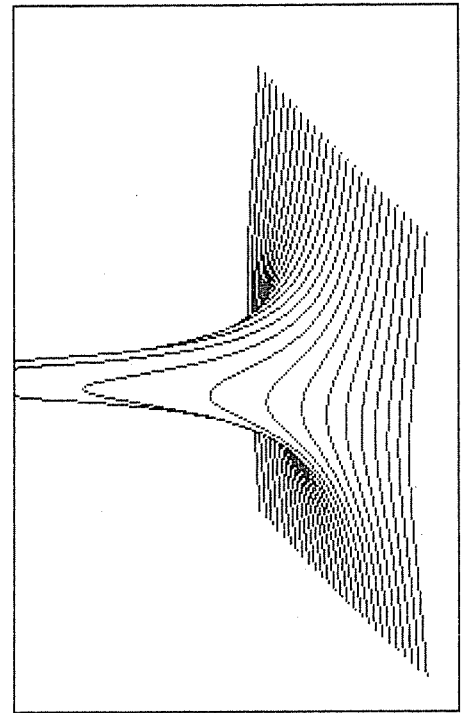


Fig. 11.112

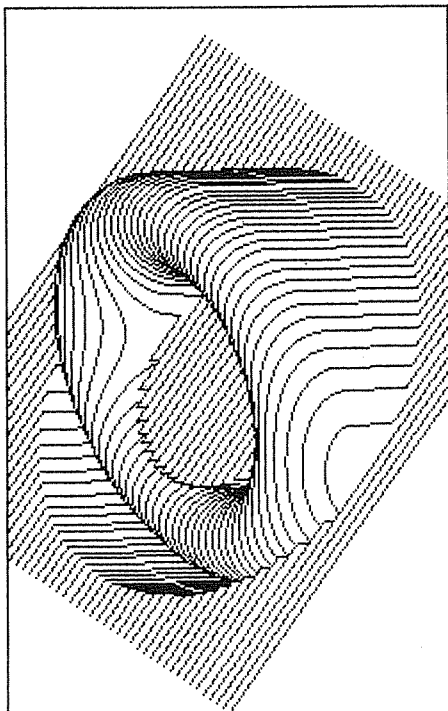


Fig. 11.113

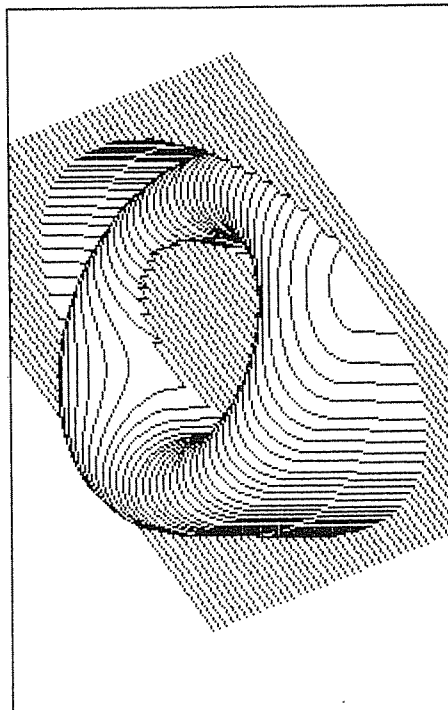
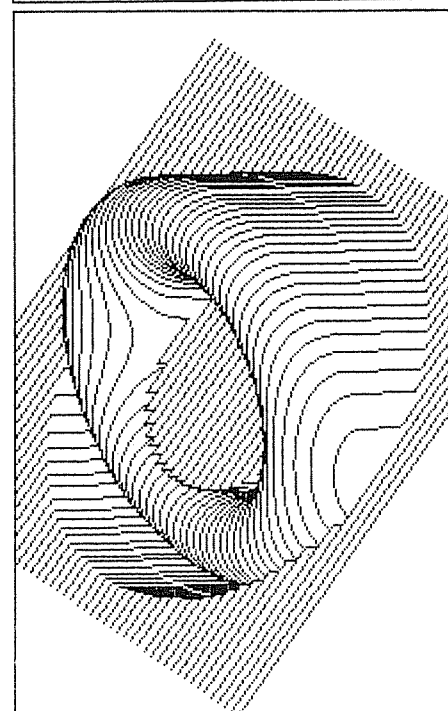
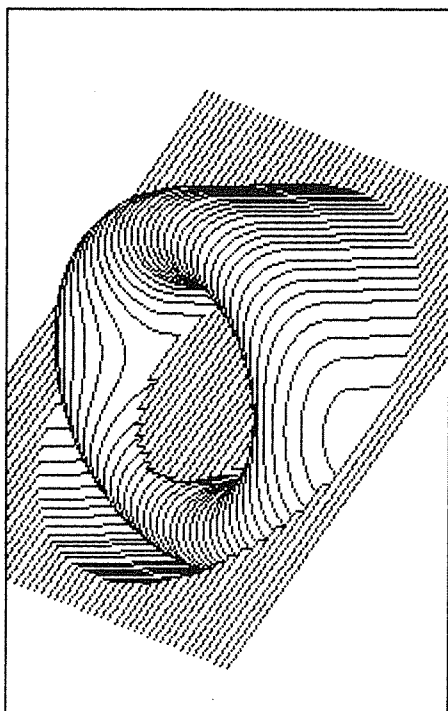


Fig. 11.114



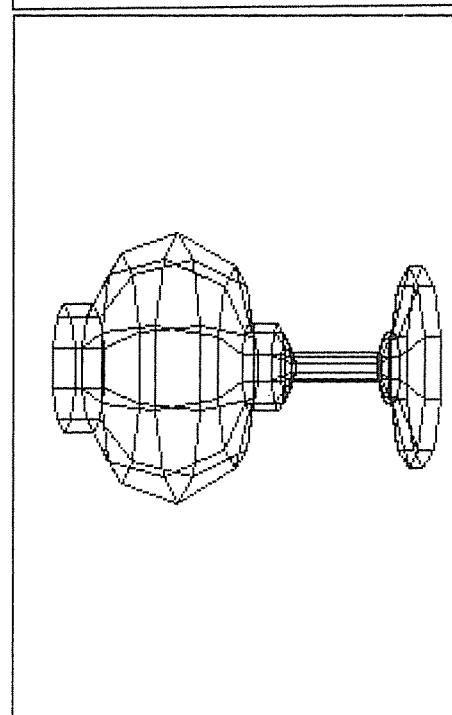
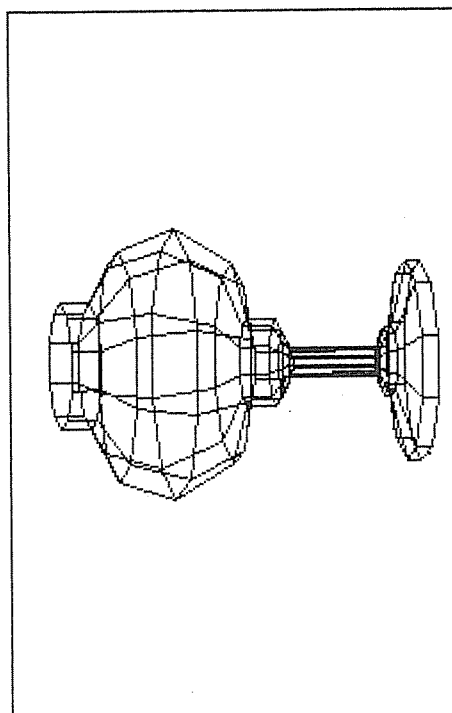


Fig. 11.115

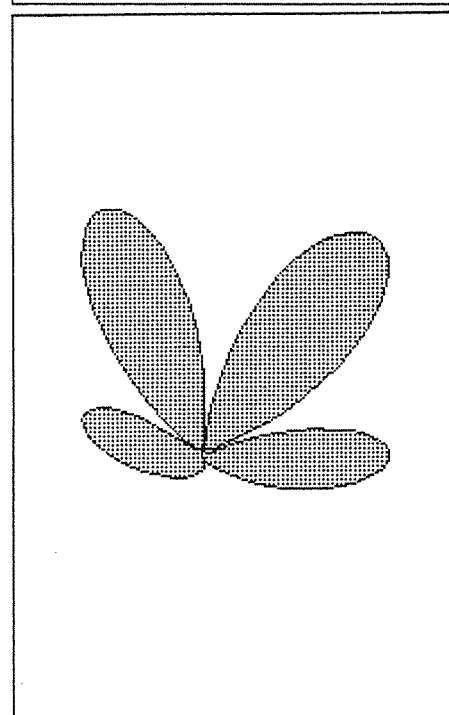
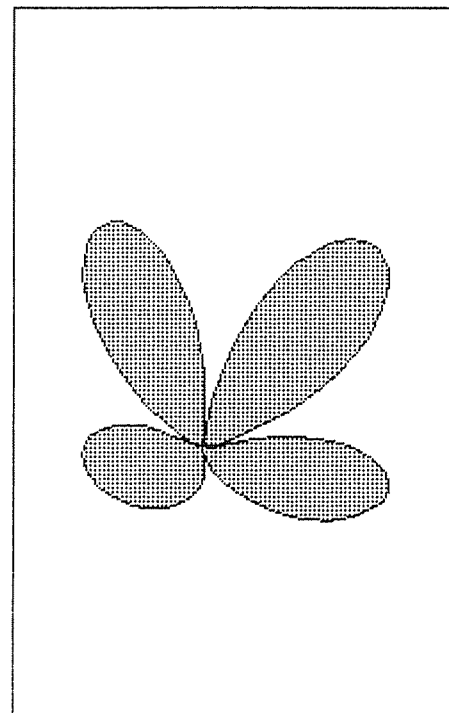


Fig. 11.116

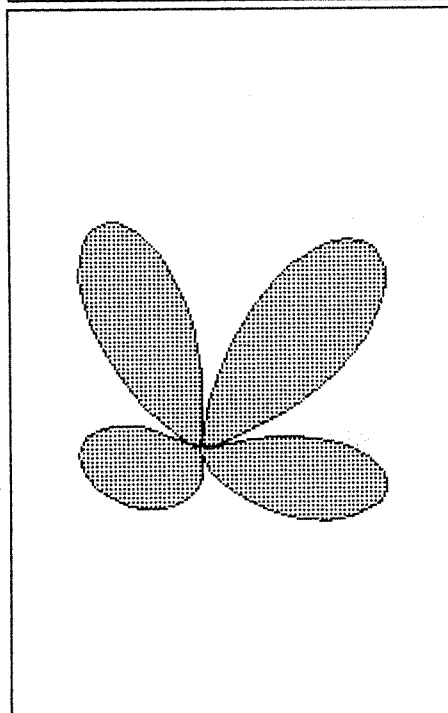
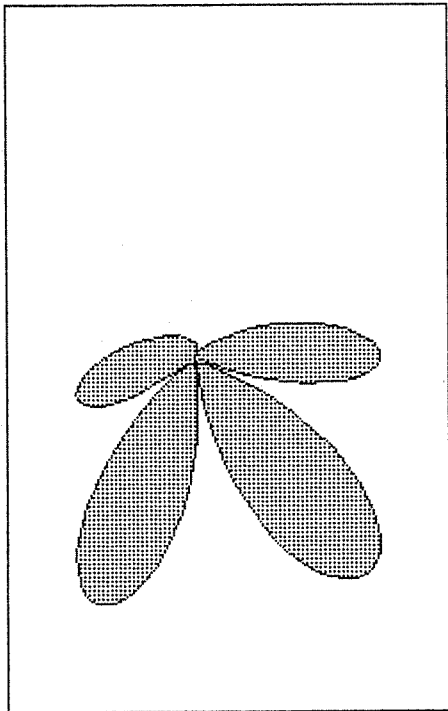


Fig. 11.117

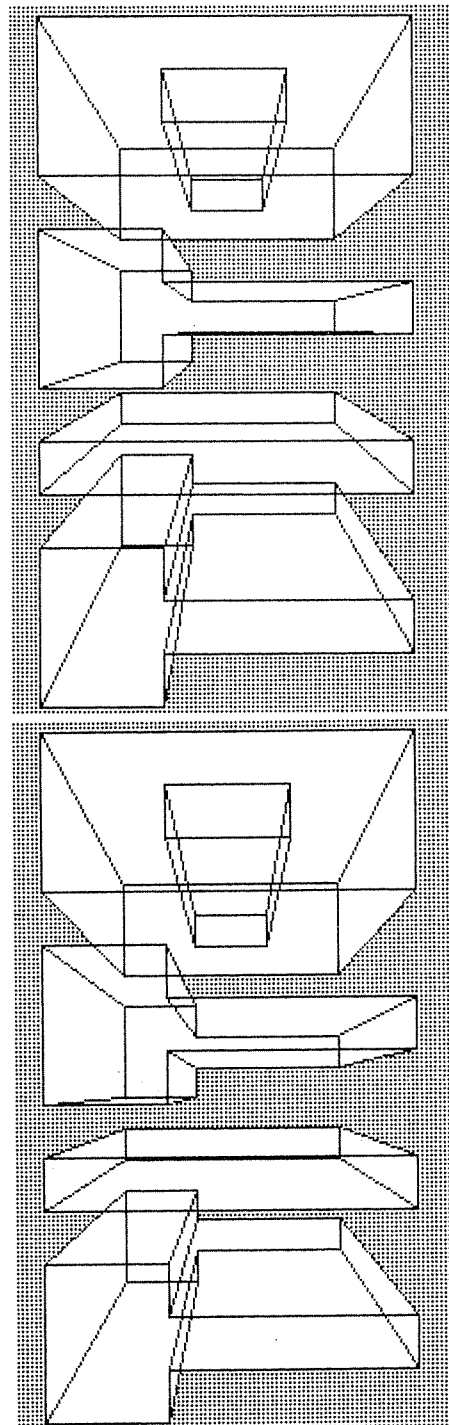


Fig. 11.118

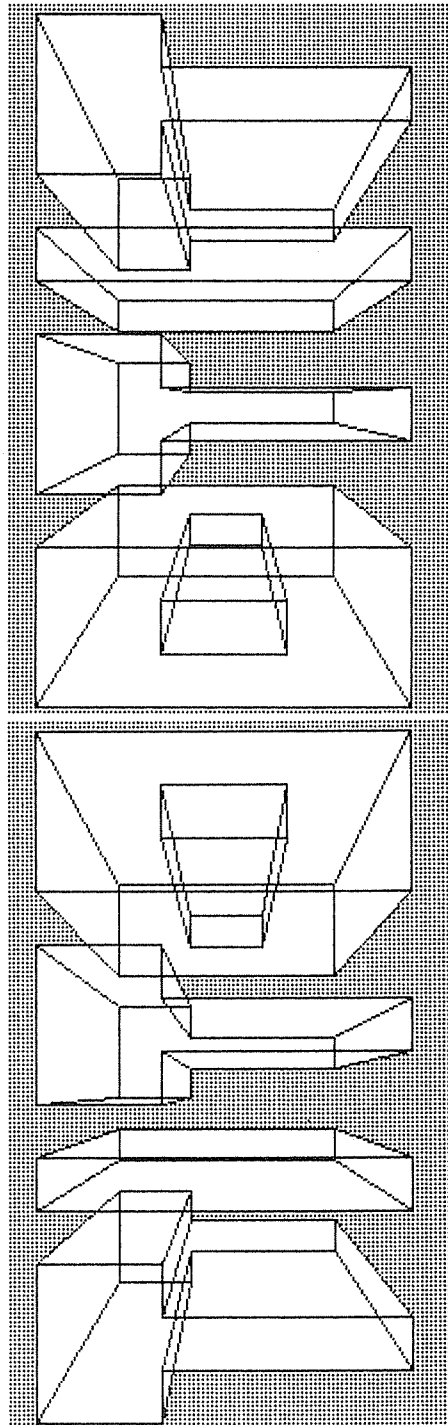


Fig. 11.119

EXERCICES

1. Une autre manière pour éliminer les lignes cachées dans les fonctions 3-D est de garder une table des minima, un index pour chaque colonne de l'écran. La fonction est tracée en partant du devant et avant qu'une ligne soit tracée, sa hauteur est comparée avec la plus haute tracée jusqu'ici dans cette colonne. Seules celles qui sont plus hautes sont tracées. Ecrire un programme utilisant cette méthode.
2. Ecrire un programme qui trace une fonction avec la grille X et Y, avec les lignes cachées supprimées et qui montre la surface du fond.
3. Les solides de révolution sont les objets générés par une forme à deux dimensions tournant autour de l'un des axes. Par exemple, une mince ellipse horizontale tournant autour de l'axe des X génère un objet semblable à un cigare. En tournant autour de l'axe des Y, elle génère un objet ressemblant à un frisbee rembourré. Générer une vue stéréoscopique du solide généré par un cercle, un carré, un octogone et une ellipse.
4. Dans tous les programmes pour tracer des fonctions 3-D, nous traversons la fonction avec des lignes horizontales, verticales ou une combinaison des deux. Ceci, bien sûr, rend les choses plus faciles, mais il n'y a pas vraiment de limite sur le type de figure que nous choisissons pour traverser la fonction, puisque toute figure dans le plan X-Y est faite de points, et ces points ont des coordonnées X et Y, avec lesquelles on peut calculer la fonction. A la figure 11.120, une fonction semblable à la croix de la figure 11.28 a été traversée par des cercles. Ecrire un programme qui trace une fonction avec cette méthode.

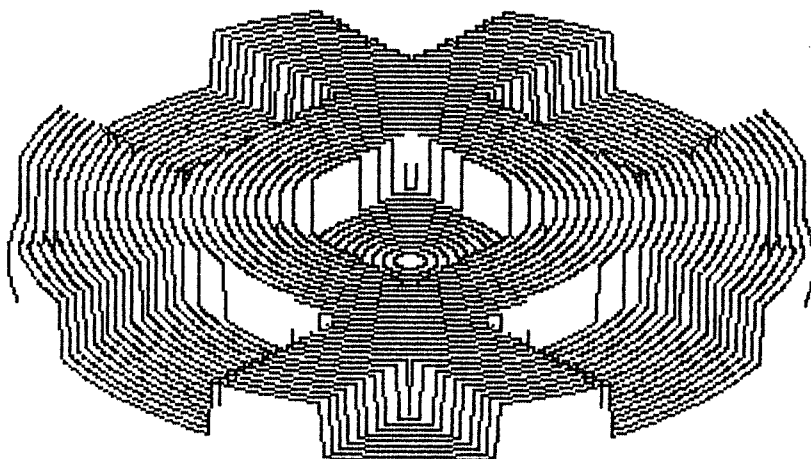


Fig. 11.120

5. En travaillant l'idée de l'exercice 4, écrire une fonction pour tracer des fonctions en les traversant avec des courbes sinusoïdales.
6. Un genre de perspective très intéressant retient d'abord notre attention en voyant les travaux de M.C. Escher, le merveilleux artiste hollandais. Comparable à l'effet d'un objectif « fish-eye », les lignes qui convergent vers le point de fuite ne sont pas droites, mais courbes, comme le montre la figure 11.121. Les courbes sont des arcs de cercles, dont les rayons sont inversement proportionnels à la distance verticale au point $Y=0$. Le cercle au point $Y=0$ a un rayon infini, c'est donc une ligne droite. Un cercle proche de $Y=0$ a un grand rayon et sa courbure est à peine remarquable. Une courbe avec un grand Y a un petit rayon et donc sa courbure est prononcée. Écrire un programme dans lequel la perspective soit calculée par cette méthode.

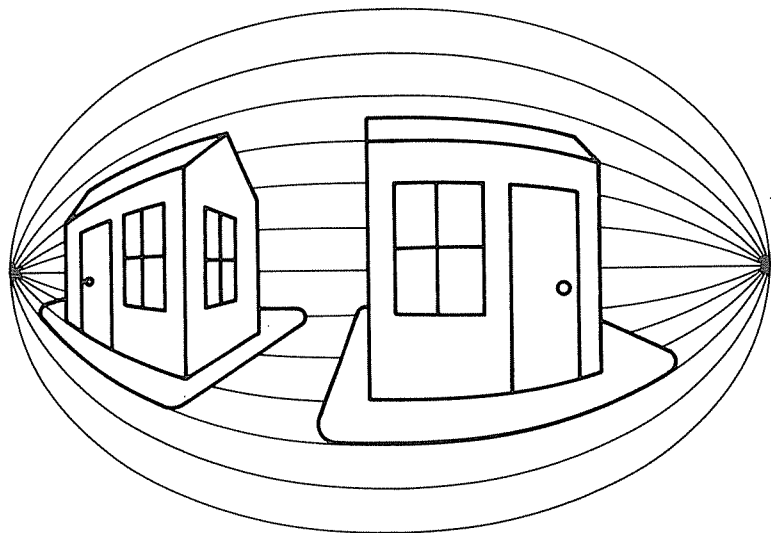


Fig. 11.121

7. Écrire un programme dans lequel les perspectives horizontale et verticale soient calculées toutes les deux en utilisant les courbes d'Escher. La figure 11.122 montre les lignes convergentes.
8. Il existe un processus avec lequel il est possible de dessiner des lettres et d'autres objets relativement simples pour produire des graphiques semblables à ceux des fonctions tridimensionnelles. La figure 11.123 montre le nom « Stella » (un nom très cher à l'auteur) en utilisant cette représentation. Notez que chaque ligne horizontale a trois possibilités : elle est en haut, en bas ou va du bas vers le haut ou vice versa. Les lettres (ou d'autres formes) sont d'abord dessinées

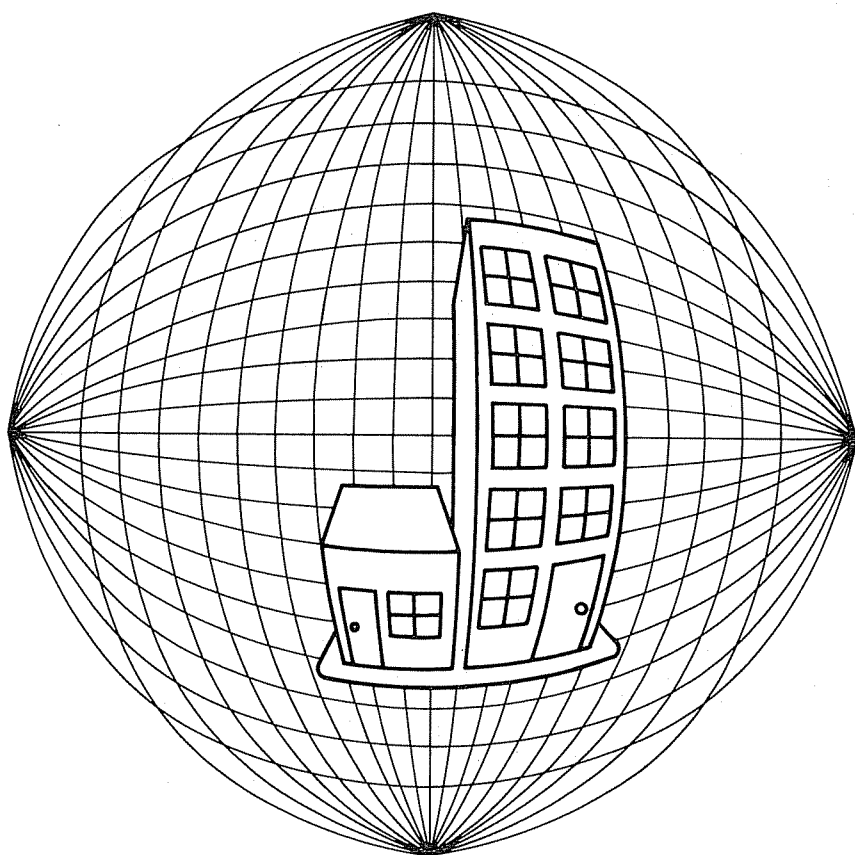


Fig. 11.122

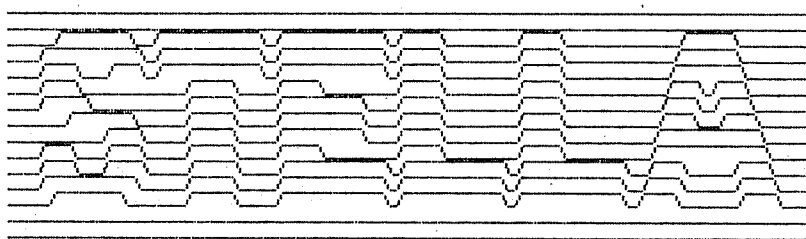


Fig. 11.123

sur un papier graphe, et chaque ligne horizontale est traversée en comptant les carrés qui sont couverts ou non par les lettres. Par exemple la troisième ligne de la figure 11.123 (à partir du haut) pourrait être décrite grossièrement en disant qu'elle a 14 unités en bas, 20 en haut, 4 en bas, 30 en haut, etc. Dans un programme pour tracer ce genre de figure, chaque fois qu'un nombre est lu, l'état doit être changé de haut à bas ou vice versa. Un code spécial (par exemple un nombre négatif) doit être utilisé pour signaler la fin de la ligne. Le programme peut être écrit juste en quelques lignes : mesurer les lettres est ce qui prend le plus de temps.

9. Ajouter la perspective au calice du programme 11.47.
10. Ecrire un programme pour tracer la figure 11.124 qui est une variation de la vague de la figure 11.22.

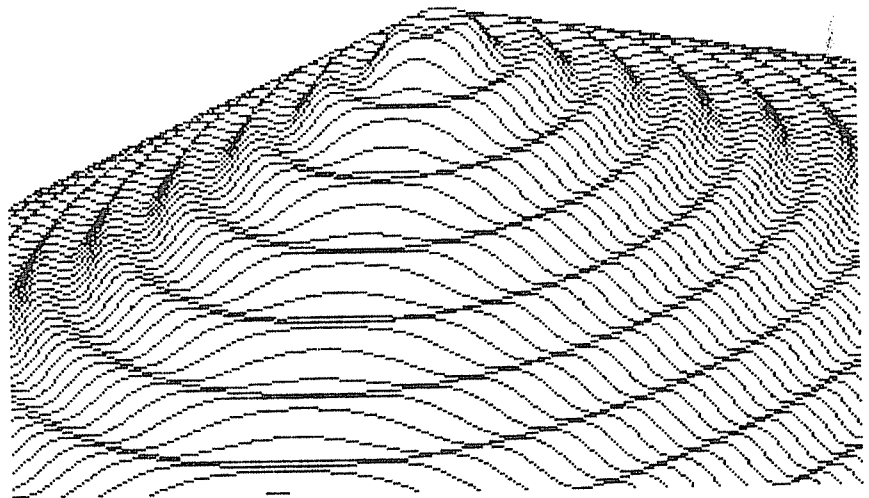


Fig. 11.124

NOTES

1. Si un des points limites d'une ligne ou les deux tombent en dehors de l'écran, quelques déformations peuvent se produire, comme il est décrit au paragraphe 2.6.3, où les méthodes pour écrêter ces lignes sont décrites.
2. Il peut y avoir plus d'un point de fuite. Pour le moment nous n'en étudierons qu'un.
3. Nous ne prenons pas la perspective en considération ici. Nous supposons que les points de fuite sont à l'infini et donc que les lignes de perspective sont parfaitement parallèles.

4. Il y a nombre de restrictions à la forme des polygones. Toutefois nous les passerons sous silence jusqu'à la fin de ce paragraphe.
5. Les maisons apparaissent toujours dans les livres sur les graphiques comme exemples pour tout : ceci parce qu'un tétraèdre est trop simple et qu'un cube est ennuyeusement symétrique. Une maison est une des figures les plus simples qui ne se répète pas quand on la tourne, tout en restant extrêmement simple.
6. Pour une explication détaillée de ceci et d'autres tris, voyez le livre « Fancy Programming with IBM PC » de Cuellar.
7. Les yeux peuvent en fait être entraînés pour voir ces figures sans dispositif externe.
8. On trouve des viseurs stéréoscopiques dans le commerce.

12

Applications de gestion

« Un bon croquis vaut mieux qu'un long discours » dit le vieil adage. Dans le cas de l'information produite par l'ordinateur, une image peut donner un sens à un ensemble de valeurs qui semblaient ne pas avoir de relation entre elles. La figure 12.1a montre une série de nombres et la figure 12.1b montre une des représentations graphiques possibles de ces valeurs. Le graphique montre nettement le comportement des nombres, tandis que les valeurs sont assez peu révélatrices par elles-mêmes.

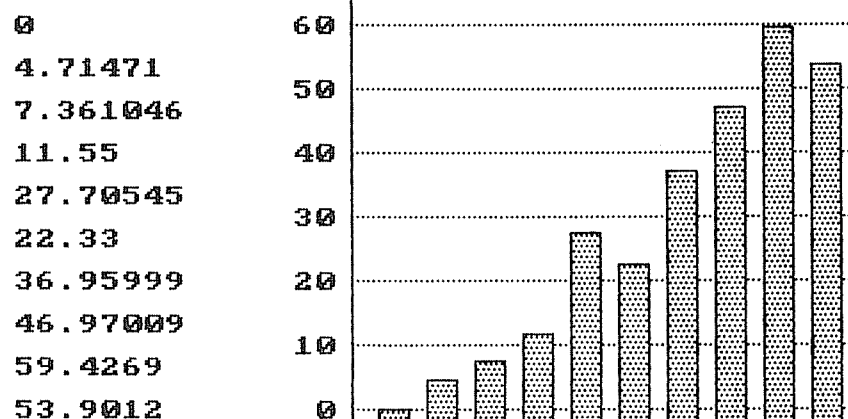


Fig. 12.1

Il y a plusieurs manières de représenter des ensembles de nombres graphiquement. Dans ce chapitre nous montrerons quelques-uns des formats les plus souvent utilisés, en prêtant plus d'attention aux méthodes pour graduer les valeurs de telle sorte que l'information soit communiquée d'une manière précise et compréhensible. Nous commencerons par une partie qui doit vous être familière maintenant : nous parlons des fonctions.

12.1 REPRESENTATION FONCTIONNELLE

Comme nous l'avons expliqué au paragraphe 1.6, une fonction est une formule ou une procédure qui affecte un nombre unique à chaque valeur donnée. La fonction Cosinus (COS en BASIC), par exemple, affecte une valeur comprise entre -1 et $+1$ à chaque nombre réel. Notez que plusieurs nombres peuvent avoir la même valeur dans une fonction ($\text{SIN}(0) = \text{SIN}(2\pi)$), mais la fonction ne donnera pas deux réponses différentes pour le même nombre. Il est donc impossible d'avoir, par exemple, $\text{SIN}(2) = A$ et $\text{SIN}(2) = B$, à moins que A et B soient identiques.

Supposez qu'on ait estimé qu'une certaine compagnie augmenterait son capital de 0,5 % par mois l'année prochaine. Etant donné un capital de départ, le programme 12.1 calcule les valeurs du capital dans les vingt-quatre prochains mois.

```
0  *** 12-1 : Calculate capital **
5  CLS
10 INPUT "Starting capital ";CAPITAL
20 FOR I=1 TO 24
30   PRINT I,CAPITAL
40   CAPITAL=CAPITAL*1.05
50 NEXT
```

PROGRAMME 12.1

Bien que les nombres reflètent très précisément la croissance prédite, un groupe de nombres ne donne pas une idée nette du comportement de l'ensemble. Nous utiliserons les valeurs produites par le programme 12.1 pour créer un graphique, mais avant de faire cela, nous devons prêter une attention spéciale à l'orientation inhabituelle de l'écran. Le 0 des coordonnées Y est situé en haut de l'écran, et les valeurs positives croissent vers le bas. Nous (Occidentaux) avons l'habitude d'assimiler de hautes valeurs avec de hauts points, donc la représentation verticale de l'écran nous semble inversée. Heureusement il y a une manière toute simple de résoudre ce problème. La plus grande valeur que l'on peut tracer est 199. Si nous retranchons les valeurs Y de 199, le graphique sera retourné la tête en bas. Par exemple zéro deviendra 199 ($199-0$), le haut de l'écran, 199 deviendra 0 ($199-199$), le bas de l'écran, et tout autre valeur comprise entre les deux sera placée à sa position correcte sur l'écran inversé.

Pour revenir à notre exemple sur la croissance de la compagnie, nous pouvons maintenant tracer les valeurs du programme 12.1 en utilisant des points. Le programme 12.2 utilise un capital de départ de 40 et trace les points qui correspondent aux 24 premiers mois.

```
0 *** 12-2 : Plot growth of company **
10 SCREEN 1:CLS
20 CAPITAL=40
30 LINE(0,0)-(0,199)
40 LINE(0,199)-(319,199)
50 FOR I=1 TO 24
60   PSET(I,199-CAPITAL)
70   CAPITAL=CAPITAL*1.05
80 NEXT
```

PROGRAMME 12.2

Les lignes 30 et 40 tracent les axes vertical et horizontal comme points de référence. Bien que les points ne donnent pas une idée de la taille du capital parce qu'il n'y a pas du tout de référence aux valeurs, la figure 12.2 montre nettement la relation entre les valeurs.

L'image peut être plus claire, s'il y a une plus grande séparation entre les points. Ceci est facilement réalisé si les coordonnées X sont multipliées par un certain facteur. Nous avons choisi de prendre 12 dans cet exemple. Si on trace une ligne au lieu d'un simple point, le graphique sera encore plus facile à comprendre. La figure 12.3 a été dessinée en changeant la ligne 60 du programme 12.2 en

```
60   LINE(I*12,199-CAPITAL)-(I*12,199)
```

Ceci nous amène directement à l'histogramme dans lequel chaque valeur est représentée par un rectangle plein. La figure 12.4 a été dessinée en changeant la ligne 60 du programme 12.2 en

```
60   LINE(I*12-5,199-CAPITAL)-(I*12+2,199),1,BF:
      LINE(I*12-5,199-CAPITAL)-(I*12+2,199),3,B
```

Dans cet exemple, les valeurs étaient calculées avec une fonction, mais bien sûr, elles auraient pu venir de n'importe où. Elles auraient pu être lues à partir du clavier, du disque ou d'un autre programme.

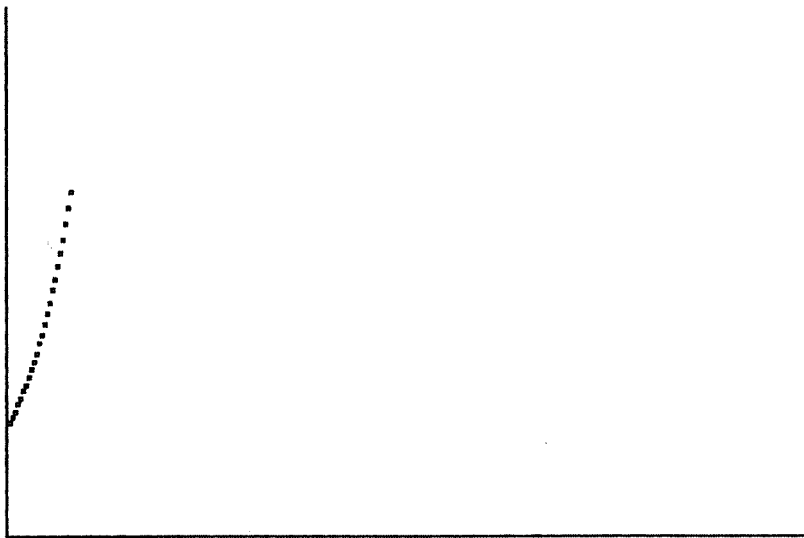


Fig. 12.2

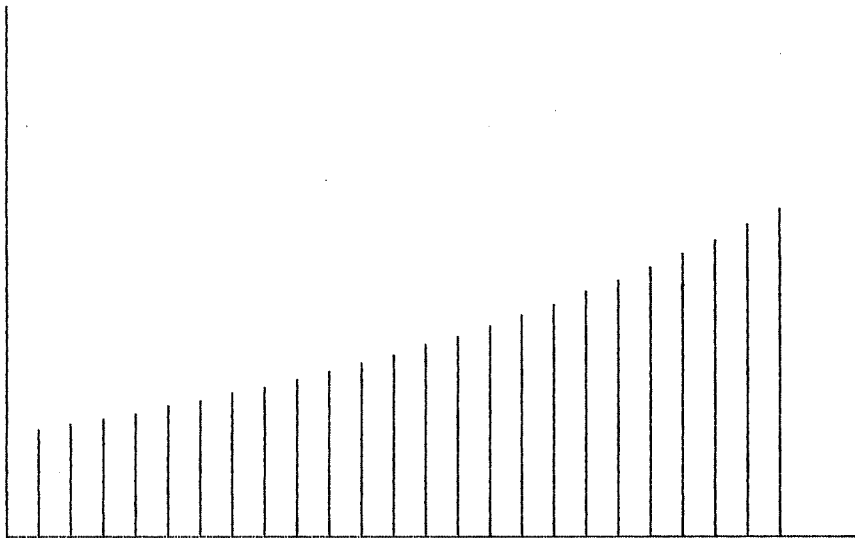


Fig. 12.3

12.2 L'ECHELLE

Les valeurs que nous avons utilisées dans les figures précédentes allaient bien pour l'écran, et à part le redressement des coordonnées Y, aucun autre calcul n'était nécessaire. Comme les valeurs qui vont être représentées

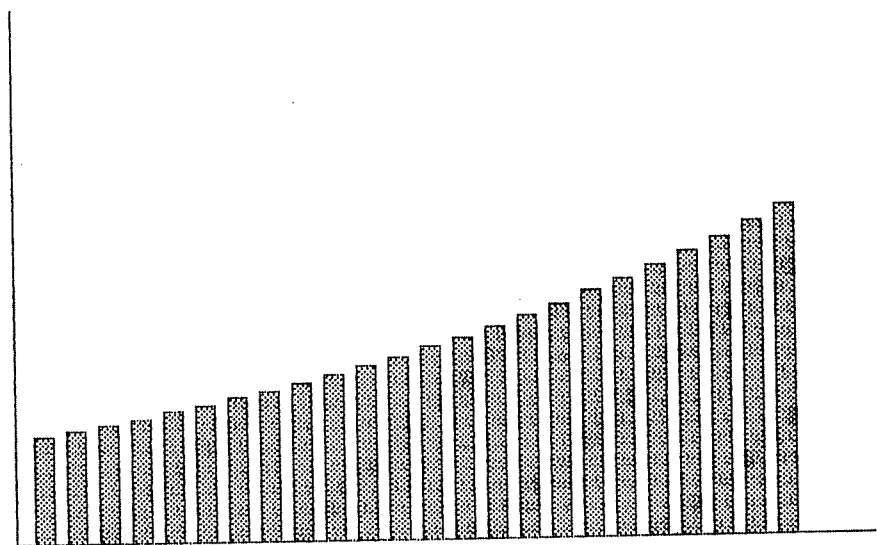


Fig. 12.4

graphiquement sont imprévisibles, il n'y a aucune façon de savoir si les valeurs vont être trop petites ou trop grandes pour l'écran. Si le capital initial du programme 12.2 avait été 400, par exemple, toutes les valeurs seraient tombées en dehors de l'écran. D'autre part, si on avait pris un capital très petit, les barres (ou les points, ou les lignes selon la représentation utilisée) auraient été si petites qu'elles n'auraient eu aucun sens.

Un graphique clair peut être produit quand les valeurs sont graduées pour convenir exactement aux proportions de l'écran. Supposez que chaque valeur soit positive, nous voulons que zéro soit au bas de l'écran et que la valeur maximum ait une hauteur de 199. Nous devons donc trouver un facteur qui, multiplié par la plus grande valeur, donnera 199, ou :

$$\text{MAX} \times \text{FACTOR} = 199$$

Une simple division est nécessaire pour calculer ce facteur :

$$\text{FACTOR} = 199 / \text{MAX}$$

Si toutes les valeurs sont multipliées par FACTOR et que les résultats sont tracés, la valeur maximum atteindra le haut de l'écran et les autres seront graduées proportionnellement. Voyons deux exemples.

Exemple 1.

Nombres : N1 = 100, N2 = 280, N3 = 310, N4 = 560
 Nombre maximum : 560

Facteur : 0,3553571 (199/560)

Les nombres sont tracés à :

N1:	35	INT(100*FACTOR)
N2:	100	INT(280*FACTOR)
N3:	110	INT(310*FACTOR)
N4:	199	INT(560*FACTOR)

Exemple 2 .

Nombres : N1 = 0,123, N2 = 0,27, N3 = 0,491, N4 = 0,3

Nombre maximum : 0,491

Facteur : 405,2953 (199/0,491)

Les nombres sont tracés à :

N1:	49	INT(0.123*FACTOR)
N2:	109	INT(0.27*FACTOR)
N3:	199	INT(0.491*FACTOR)
N4:	121	INT(0.3*FACTOR)

Le programme 12.3 créé l'histogramme qui correspond aux 12 valeurs rentrées dans la boucle FOR aux lignes 80 à 110. La figure 12.5 montre le graphique produit.

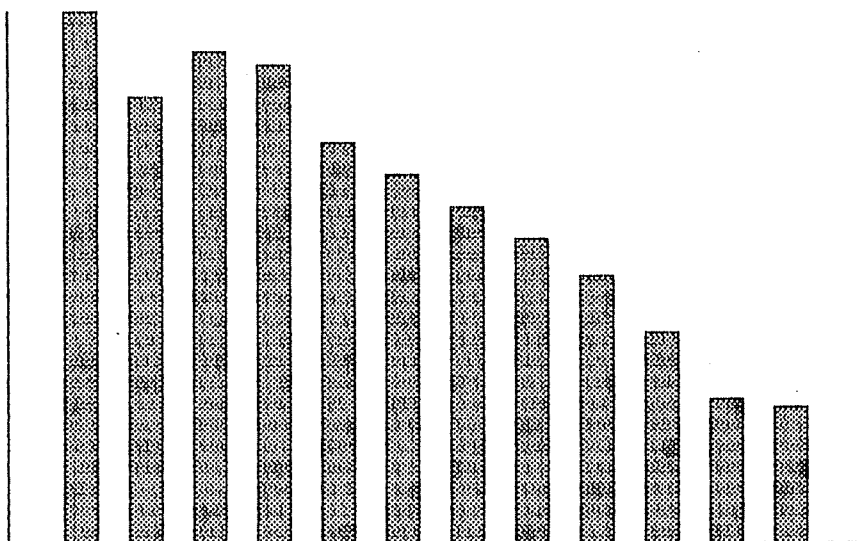


Fig. 12.5

Essayez-le avec des nombres positifs très grands et très petits pour voir comment chaque barre est proportionnelle à la plus grande valeur.

```

0 ' ** 12-3 : Bar chart **
10 SCREEN 1:COLOR 0,0:CLS
20 DIM S(12)
30 MAX=1
40 FOR I=1 TO 12
50   PRINT I;:INPUT S(I)
60   IF S(I)>S(MAX)
      THEN
        MAX=I
70 NEXT
80 CLS
90 LINE(0,0)-(0,199)
100 LINE(0,199)-(319,199)
110 D=S(MAX)-S(MIN):FAC=199/D
120 FOR I=1 TO 12
130   X=5+I*24
140   Y=199-(S(I)-S(MIN))*FAC
150   LINE(X,Y)-(X+13,199),2,BF
160 NEXT
170 W$=INPUT$(1)

```

PROGRAMME 12.3

Notez que seules les valeurs positives sont graduées correctement. Au paragraphe 12.4 nous corrigerons ce défaut, plus un autre problème qui peut être causé par certains types spéciaux de données.

12.3 L'ANNOTATION DES AXES

Il est important d'indiquer ce à quoi correspond chaque barre (un mois, un département, etc.). Comme l'échelle prend de la place, il est également nécessaire d'indiquer ce que signifient les différentes hauteurs. Les graphiques créés par les programmes dans les paragraphes précédents étaient utiles parce qu'ils montraient les relations entre les différentes valeurs, mais ils ne disaient pas si ces valeurs étaient grandes ou petites.

Nous commencerons par annoter l'axe des X dans lequel chaque barre représentera un des douze mois de l'année. Nous n'utiliserons pas d'autre notation mais bien sûr on peut utiliser n'importe quelle chaîne de caractères.

On utilise d'habitude trois lettres pour indiquer les mois. S'ils sont imprimés horizontalement, avec un seul espace entre eux, on aura besoin de 47 positions. Si on utilise la haute résolution on aura suffisamment de place, mais pas en moyenne résolution. La moyenne résolution demande un arrangement différent, c'est-à-dire le placement des noms verticalement comme le montre la figure 12.6.

Le programme 12.4 imprime les noms des mois de cette manière.

J	F	M	A	M	J	J	A	S	O	N	D
A	E	A	P	A	U	U	U	E	C	O	E
N	B	R	R	Y	N	L	G	P	T	V	C

Fig. 12.6

```

0  ** 12-4 : Arranges names of months **
10 SCREEN 1:CLS
20 DIM M$(12)
30 FOR I=1 TO 12
40   READ M$(I)
50 NEXT
60 DATA JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC
70 FOR I=1 TO 12
80   FOR J=1 TO 3
90     LOCATE 22+J,2+3*I:PRINT MID$(M$(I),J,1);
100  NEXT
110 NEXT

```

PROGRAMME 12.4

Du fait que les lettres occupent une partie du bas de l'écran, on ne peut plus placer l'axe des X sur la 199^{ème} ligne. La valeur la plus grande pour laquelle il n'y aura pas d'interférence entre les caractères et la ligne sera 174. Nous prendrons 172.

Annotons maintenant l'axe des Y. Ayant décidé où nous placerons l'axe des X, le zéro doit être placé à la même hauteur (rappelez-vous que nous ne considérons que des valeurs positives). Nous savons également que la valeur maximum va être proche du haut de l'écran, aussi nous pouvons mettre cette valeur à la position de caractère (1,1). Toutefois ces nombres doivent être manipulés avec soin parce que chaque position de caractère est capitale lorsqu'il y a plusieurs valeurs à tracer. Les nombres sont toujours imprimés précédés par un blanc s'ils sont positifs et par le signe moins s'ils sont négatifs, on ajoute également un blanc à la fin de chaque nombre ¹. Comme nous n'avons affaire qu'à des nombres positifs, nous voulons nous débarrasser de ces blancs inutiles. Ceci est réalisé facilement avec la fonction RIGHT\$, comme le montre la fonction 12.1.

```
10 DEF FN NUM$(X)=RIGHT$(STR$(X),LEN(STR$(X))-1) (12.1)
```

La ligne 220 du programme 12.5 imprime à l'emplacement (1,1) le maximum de douze valeurs rentrées des lignes 80 à 110 et la ligne 230 imprime « 0 » à la même hauteur sur l'axe des X. L'axe vertical est dessiné à la ligne 140 et l'axe horizontal à la ligne 150. Notez que ces deux axes ont

été placés de telle sorte qu'il y ait assez de place libre pour les nombres et les noms des mois.

```

0 '** 12-5 : Scaled with maximum **
10 DEF FN NUM$(X)=
    RIGHT$(STR$(X),LEN(STR$(X))-1)
20 SCREEN 1:COLOR 0,0:CLS
30 DIM M$(12),S(12)
40 FOR I=1 TO 12
50   READ M$(I)
60 NEXT
70 DATA Jan, Feb, Mar, Apr, May, Jun,
    Jul, Aug, Sep, Oct, Nov, Dec
80 MAX=1
90 FOR I=1 TO 12
100  PRINT M$(I);:INPUT S(I)
110  IF S(I)>S(MAX)
    THEN
      MAX=I
120 NEXT
130 CLS
140 LINE(20,0)-(20,170)
150 LINE(20,170)-(319,170)
160 FOR I=1 TO 12
170   FOR J=1 TO 3
180     LOCATE 22+J,2+3*I
190     PRINT MID$(M$(I),J,1);
200   NEXT
210 NEXT
220 LOCATE 1,1:PRINT FN NUM$(S(MAX));
230 LOCATE 22,1:PRINT"0"
235 LOCATE 22/2,1:
    PRINT FN NUM$(INT(S(MAX)/2));
240 FAC=160/S(MAX)
250 FOR I=1 TO 12
260   X=5+I*24
270   Y=165-S(I)*FAC
280   LINE(X,Y)-(X+13,170),2,BF:
    LINE(X,Y)-(X+13,170),3,B
290 NEXT
300 W$=INPUT$(1)

```

PROGRAMME 12.5

On peut utiliser plus de divisions entre le zéro et le nombre maximum. Pour placer une division intermédiaire, ajoutez le programme 12.6 au programme 12.5.

```

225 LOCATE 22/2,1:PRINT FN NUM$(CINT(S(MAX)/2));

```

PROGRAMME 12.6

Le curseur de caractères est positionné à mi-chemin entre 0 et le maximum, et le nombre imprimé ici est le maximum divisé par 2. Pour dessiner la figure 12.7, une ligne en pointillé a été ajoutée au milieu de l'axe vertical en $Y = 86$ (puisque le maximum est 172).

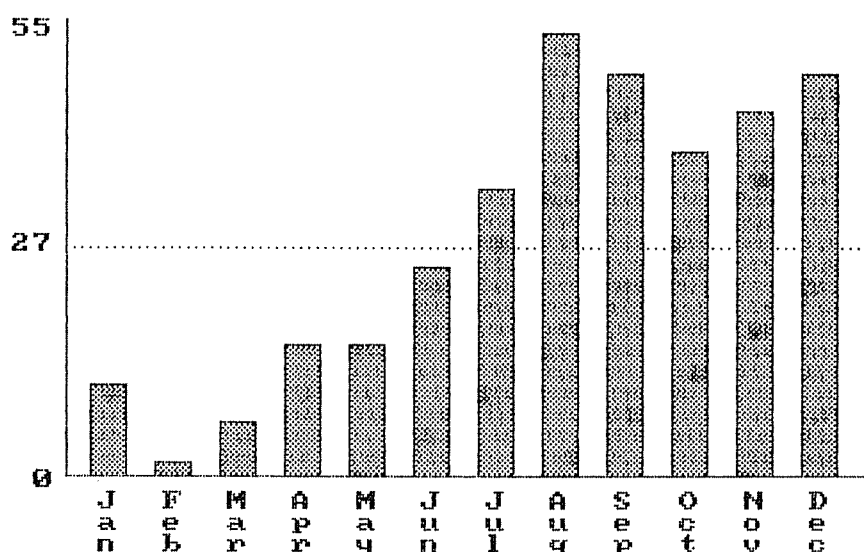


Fig. 12.7

Le programme 12.7 montre comment rajouter deux références numériques supplémentaires à l'axe vertical.

```
236 LOCATE 22/4,1:PRINT FN NUM$(CINT(3*S(MAX)/4));
237 LOCATE 3*22/4,1:PRINT FN NUM$(CINT(S(MAX)/4));
```

PROGRAMME 12.7

Si on connaît l'ordre de grandeur des nombres à l'avance, c'est une bonne idée que d'annoter l'axe numérique toutes les dix ou cent unités, comme le montre la figure 12.8.

Quand les nombres sont grands et que l'espace est étroit, un bon compromis consiste à ne faire apparaître que les deux ou trois chiffres de gauche significatifs des nombres, une condition qui doit être signalée. A la figure 12.9 les unités apparaissent en dessous du titre principal.

12.4 GRADUATION AVEC LE MINIMUM

Pour développer la méthode de graduation du paragraphe 12.2, nous supposons que toutes les valeurs étaient positives. Si on utilise des valeurs négatives avec cette technique, elles ne seront pas tracées et le graphique ne montrera pas l'information correctement. Une autre situation dans laquelle cette méthode peut produire des graphiques inadéquats c'est quand les nombres sont tous très voisins les uns des autres avec un maxi-

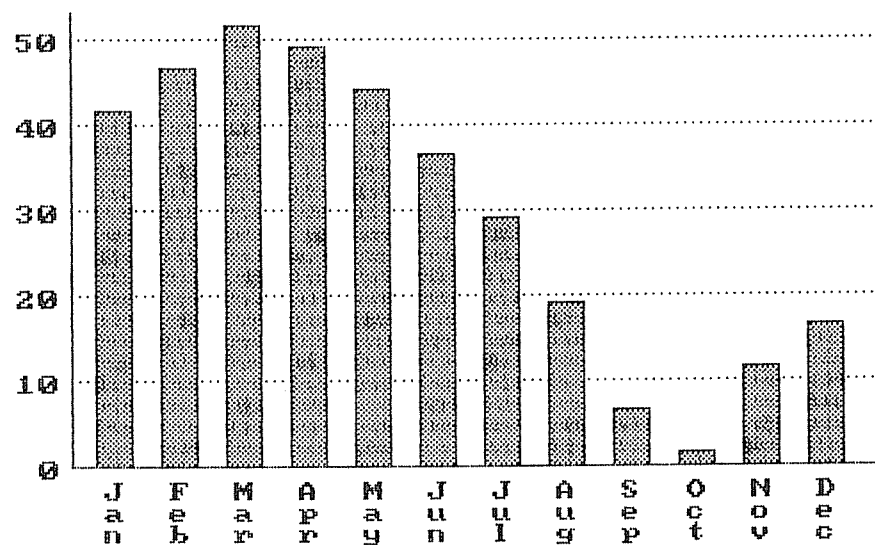


Fig. 12.8

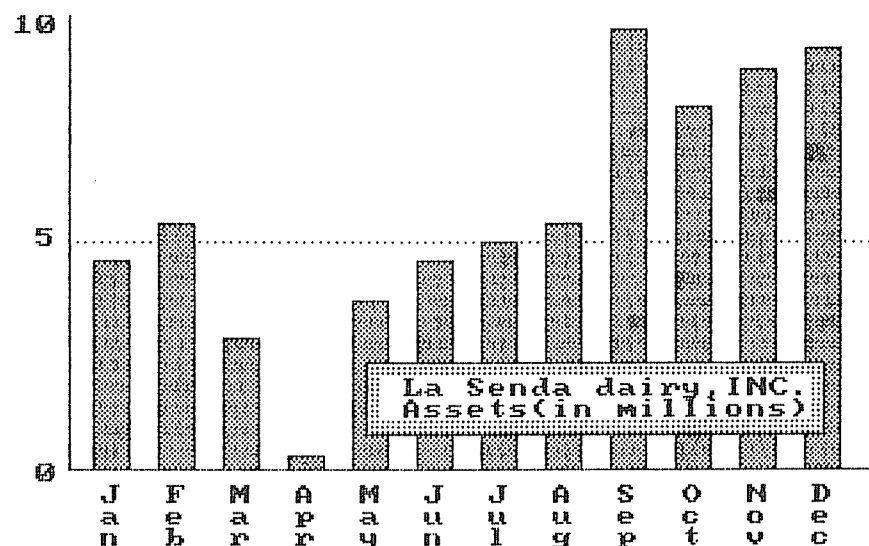


Fig. 12.9

num très grand ou très petit. Si par exemple le maximum est 150,001 des nombres comme 150,000 et 149,998 ne refléteront pas une grande différence. La figure 12.10 montre un histogramme avec des valeurs comprises entre 980 et 992.

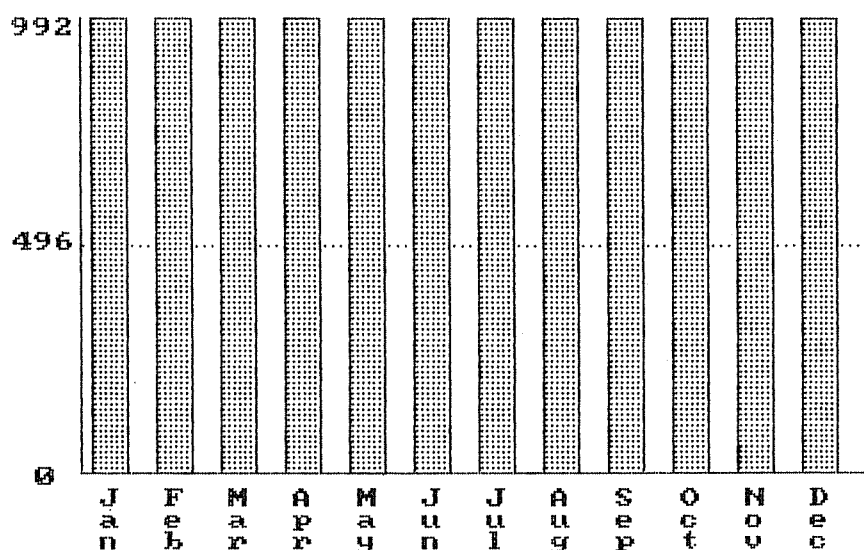


Fig. 12.10

Nous allons développer ici une méthode pour échelonner les valeurs en prenant en considération le minimum et le maximum. L'idée est simplement de faire correspondre la plus petite valeur au bas de l'écran (au-dessus des annotations de l'axe des X) et la valeur maximum au haut de l'écran et de graduer toutes les valeurs intermédiaires en conséquence ².

Nous vous rappelons qu'au paragraphe 12.2 nous avons vu que pour déterminer le facteur d'échelle, on divisait la hauteur de l'écran (valeur maximum : 199) par le nombre maximum. Tout nombre positif multiplié par ce facteur était gradué et le nombre maximum devenait 199. Comme l'axe des X est maintenant à la ligne 172 (à cause des annotations), nous déterminerons le facteur en divisant 172 par le nombre maximum. Pour déterminer le facteur d'échelle en considérant le minimum et en permettant des nombres négatifs, nous utiliserons une méthode semblable, que nous expliquerons avec un exemple. Supposez que la plus petite valeur soit -10 et la plus grande 300. Si nous retranchons -10 de tous les nombres (donc ajoutant 10), le minimum deviendra 0 et le maximum 310. Nous pouvons maintenant appliquer notre méthode pour déterminer le facteur. En supposant que le maximum soit MAX et le minimum MIN, les étapes à suivre sont les suivantes :

1. Trouver la différence des deux nombres ($D = \text{MAX} - \text{MIN}$)
2. Trouver le facteur de la différence ($\text{FACTOR} = 172/D$)
3. Soustraire le minimum de chaque nombre.
4. Multiplier chaque nombre par le facteur.

Exemple 1.

Nombres : N1 = 300, N2 = 304, N3 = 307, N4 = 312

Minimum : 300

Maximum : 312

Différence : 12 (312-300)

Facteur : 14,33333 (172/300)

Les nombres sont tracés à :

N1:	0	INT((300-300)*FACTOR)
N2:	57	INT((304-300)*FACTOR)
N3:	100	INT((307-300)*FACTOR)
N4:	172	INT((312-300)*FACTOR)

Exemple 2.

Nombres : N1 = -0,8, N2 = -0,2, N3 = 0,34, N4 = 0,98

Minimum : -0,8

Maximum : 0,98

Différence : 1,78 (0,98-(-0,8))

Facteur : 96,62921 (172/1,78)

Les nombres sont tracés à :

N1:	0	INT((-0.8-(-0.8))*FACTOR)
N2:	58	INT((-0.2-(-0.8))*FACTOR)
N3:	110	INT((0.34-(-0.8))*FACTOR)
N4:	172	INT((0.98-(-0.8))*FACTOR)

Le programme 12.8 dessine l'histogramme des douze valeurs rentrées au clavier, prenant en considération le minimum et le maximum. La figure 12.11 montre un des histogrammes qui peut être produit avec le programme 12.8.

```
10 *** 12-8 : Bar chart with min,max **
20 SCREEN 1:CLS
30 DEF FN NUM$(X)=
    RIGHT$(STR$(X),LEN(STR$(X))-1)
40 DIM M$(12),S(12)
50 FOR I=1 TO 12
60   READ M$(I)
70 NEXT
80 DATA JAN,FEB,MAR,APR,MAY,JUN,
    JUL,AUG,SEP,OCT,NOV,DEC
```



```

90 MAX=1:MIN=1
100 FOR I=1 TO 12
110   PRINT M$(I);:INPUT S(I)
120   IF S(I)>S(MAX)
      THEN
        MAX=I
130   IF S(I)<S(MIN)
      THEN
        MIN=I
140 NEXT
150 CLS
160 FOR I=1 TO 12
170   FOR J=1 TO 3
180     LOCATE 22+J,2+3*I:
      PRINT MID$(M$(I),J,1);
190   NEXT
200 NEXT
210 LINE(27,0)-(27,172)
220 LINE(27,172)-(319,172)
225 FOR I=27 TO 319 STEP 3:
      PSET(I,86):
    NEXT
230 LOCATE 22,1:
      PRINT FN NUM$(INT(S(MIN)));
240 LOCATE 1,1:
      PRINT FN NUM$(S(MAX));
250 LOCATE 22/2,1:
      PRINT FN NUM$(INT((S(MIN)+S(MAX))/2));
260 FACTOR=170/(S(MAX)-S(MIN))
270 FOR I=1 TO 12
280   HEIGHT=((S(I)-S(MIN))*FACTOR)
290   LINE(I*24+5,170-HEIGHT)-(I*24+17,172),1,BF
295   LINE(I*24+5,170-HEIGHT)-(I*24+17,172),3,B
300 NEXT
2800 W$=INPUT$(1)

```

PROGRAMME 12.8

Essayez de dessiner le même histogramme avec des valeurs comprises entre 67 et 74 avec le programme 12.3.

12.5 HISTOGRAMMES SIMULTANES

Il est souvent commode de comparer deux ensembles de données. Par exemple pour comparer les ventes d'une année avec celles d'une autre on voudra tracer les valeurs de tous les deux mois correspondants, côte à côte, pour faciliter les comparaisons.

Au programme 12.9, les deux valeurs de chaque mois (première année, deuxième année par exemple) sont rentrées dans les tableaux S et T aux lignes 80 à 140. La technique pour trouver le maximum est légèrement différente parce que le programme doit trouver le maximum des deux années. Si les blocs pour les mêmes mois sont tracés légèrement décalés l'un par rapport à l'autre, il sera possible de les différencier clairement.

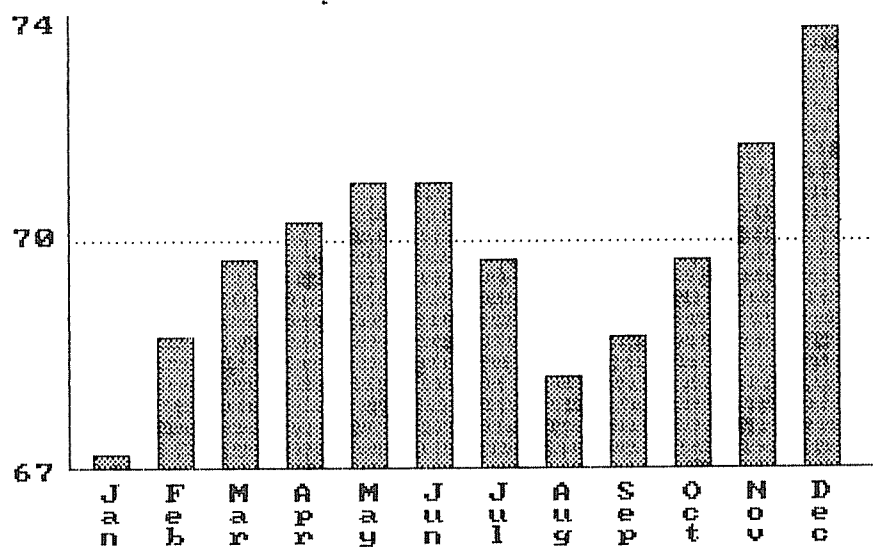


Fig. 12.11

Dans le programme 12.9 les blocs de la première année sont tracés entre les points $(X + 3, Y_T) - (X + 16, 170)$ et ceux de la seconde année entre les points $(X, Y_S) - (X + 13, 170)$. La figure 12.12 montre le graphique résultant.

```

0  ** 12-9 : Simultaneous chart **
10 SCREEN 1:COLOR 0,0:CLS
20 DIM M$(12),S(12),T(12)
30 FOR I=1 TO 12
40   READ M$(I)
50 NEXT
60 DATA Jan, Feb, Mar, Apr, May, Jun,
    Jul, Aug, Sep, Oct, Nov, Dec
70 MINS=1:MAXS=1:MINT=1:MAXT=1
80 FOR I=1 TO 12
90   PRINT M$(I);:INPUT S(I),T(I)
100  IF T(I)<T(MINT)
    THEN
      MINT=I
110  IF S(I)<S(MINS)
    THEN
      MINS=I
120  IF S(I)>S(MAXS)
    THEN
      MAXS=I
130  IF T(I)>T(MAXT)
    THEN
      MAXT=I
140 NEXT
150 CLS
160 LINE(20,0)-(20,170)
170 LINE(20,170)-(319,170)

```

```

180 FOR I=1 TO 12
190   FOR J=1 TO 3
200     LOCATE 22+J,2+3*I
210     PRINT MID$(M$(I),J,1);
220   NEXT
230 NEXT
240 DS=S(MAXS)-S(MINS):FACS=160/DS
250 DT=T(MAXT)-T(MINT):FACT=160/DT
260 FOR I=1 TO 12
270   X=5+I*24
280   YT=165-(T(I)-T(MINT))*FACT
290   YS=165-(S(I)-S(MINS))*FACS
300   LINE(X+3,YT)-(X+16,170),2,BF
310   LINE(X+3,YT)-(X+16,170),3,B
320   LINE(X,YS)-(X+13,170),1,BF
330   LINE(X,YS)-(X+13,170),3,B
340 NEXT
350 W$=INPUT$(1)

```

PROGRAMME 12.9

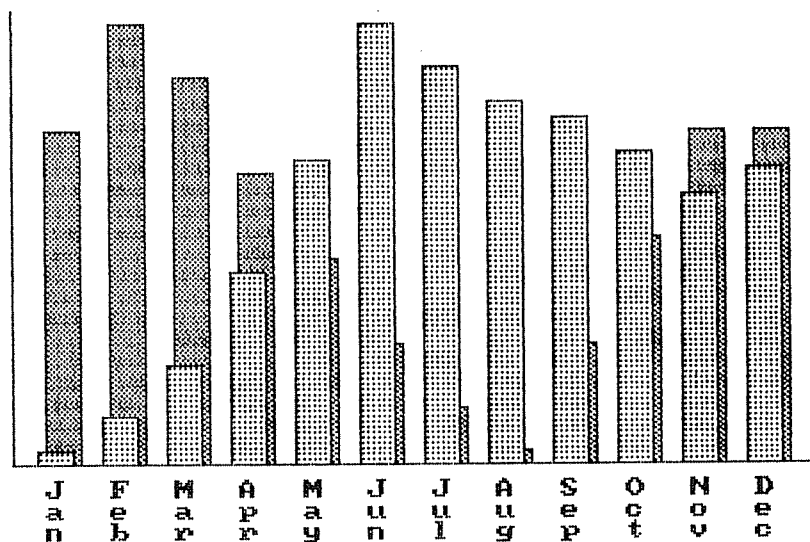


Fig. 12.12

12.6 BARRES PLEINES

Des barres pleines sont souvent préférables pour mettre en évidence des blocs en couleur. Pour afficher ces blocs, dessinez simplement la barre deux fois, en augmentant la coordonnée X et en diminuant Y, connectez les coins et colorez les murs avec une couleur voulue. Le programme 12.10 illustre cette technique, et la figure 12.13 montre le graphique réalisé.

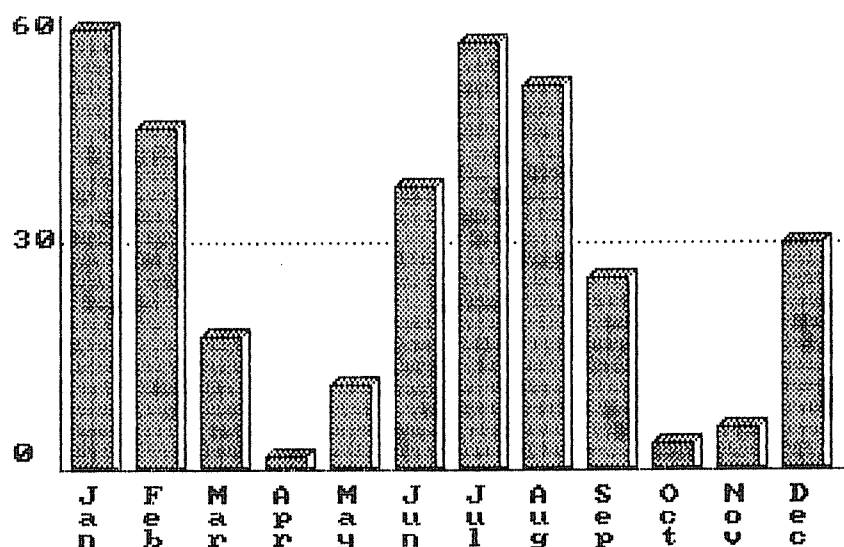


Fig. 12.13

```

0 '** 12-10 : Solid bars **
5 DEF FN S$(X)=LEFT$(STR$(X),3)
10 SCREEN 1:COLOR 0,0:CLS
20 DIM M$(12),S(12),T(12)
30 FOR I=1 TO 12
40   READ M$(I)
50 NEXT
60 DATA Jan, Feb, Mar, Apr, May, Jun,
   Jul, Aug, Sep, Oct, Nov, Dec
65 MIN=1:MAX=1
70 FOR I=1 TO 12
80   PRINT M$(I);:INPUT S(I)
85   IF S(I)<S(MIN)
      THEN
        MIN=I
87   IF S(I)>S(MAX)
      THEN
        MAX=I
90 NEXT
100 CLS
120 LINE(25,0)-(25,170)
130 LINE(25,170)-(319,170)
140 FOR I=1 TO 12
150   FOR J=1 TO 3
160     LOCATE 22+J,2+3*I
170     PRINT MID$(M$(I),J,1);
180   NEXT
190 NEXT
191 LOCATE 1,1
192 PRINT FN S$(S(MAX));
193 V=INT((S(MAX)+S(MIN))/2)
194 LOCATE 11,1:PRINT FN S$(V);
195 LOCATE 21,1:PRINT FN S$(S(MIN));

```

```

196 FOR X=26 TO 319 STEP 3
197   PSET(X,85)
198 NEXT
200 D=S(MAX)-S(MIN):FAC=160/D
1000 FOR I=1 TO 12
1010   XTEMP=5+I*24
1015   YS=165-(S(I)-S(MIN))*FAC
1020   X=XTEMP:Y=YS:FC=2:BC=3:
       GOSUB 20000
1030 NEXT
1040 W#=INPUT$(1):END
20000 'Draw a solid bar
20005 LINE(X,Y)-(X+15,169),FC,BF:
       LINE(X,Y)-(X+15,169),BC,B
20010 LINE(X+3,Y-3)-(X+18,Y-3),BC
20020 LINE(X,Y)-(X+3,Y-3),BC:
       LINE(X+15,Y)-(X+18,Y-3),BC
20030 PAINT(X+2,Y-1),FC,BC
20040 LINE(X+18,Y-3)-(X+18,169),BC
20050 LINE(X,169)-(X+18,169),BC
20054 PAINT(X+17,Y-1),2,BC
20055 PAINT(X+17,Y-1),0,BC
20060 RETURN

```

PROGRAMME 12.10

Le programme 12.11 calcule les blocs simultanés de la figure 12.14.

```

0 '** 12-11 : Double solid bar **
10 SCREEN 1:COLOR 0,0:CLS
20 DIM M$(12),S(12),T(12)
30 FOR I=1 TO 12
40   READ M$(I)
50 NEXT
60 DATA Jan, Feb, Mar, Apr, May, Jun,
       Jul, Aug, Sep, Oct, Nov, Dec
70 MINS=1:MAXS=1:MINT=1:MAXT=1
80 FOR I=1 TO 12
90   PRINT M$(I);:INPUT S(I),T(I)
100  IF T(I)<T(MINT)
       THEN
           MINT=I
110  IF S(I)<S(MINS)
       THEN
           MINS=I
120  IF S(I)>S(MAXS)
       THEN
           MAXS=I
130  IF T(I)>T(MAXT)
       THEN
           MAXT=I
140 NEXT
150 CLS
160 LINE(20,0)-(20,170)
170 LINE(20,170)-(319,170)
180 FOR I=1 TO 12
190   FOR J=1 TO 3

```

```

200     LOCATE 22+J,2+3*I
210     PRINT MID$(M$(I),J,1);
220     NEXT
230 NEXT
240 LOCATE 1,1
250 DS=S(MAXS)-S(MINS):FACS=160/DS
260 DT=T(MAXT)-T(MINT):FACT=160/DT
270 FOR I=1 TO 12
280     XTEMP=5+I*24
290     YT=165-(T(I)-T(MINT))*FACT
300     YS=165-(S(I)-S(MINS))*FACS
310     X=XTEMP+6:Y=YT:GOSUB 1000
320     X=XTEMP:Y=YS:GOSUB 1000
330 NEXT
340 W$=INPUT$(1):END
1000 'Plot solid bars
1010 FOR K=0 TO 3
1020     LINE(X+K,Y-K)-(X+10+K,170),0,BF
1030 NEXT
1040 LINE(X,Y)-(X+10,170),2,BF:
      LINE(X,Y)-(X+10,170),3,B
1050 LINE(X+3,Y-3)-(X+13,Y-3),3
1060 LINE(X,Y)-(X+3,Y-3),3:
      LINE(X+10,Y)-(X+13,Y-3),3
1070 PAINT(X+2,Y-1),3,3
1080 LINE(X+13,Y-3)-(X+13,169),3
1090 LINE(X,170)-(X+13,170),3
1100 PAINT(X+12,168),0,3
1110 RETURN

```

PROGRAMME 12.11

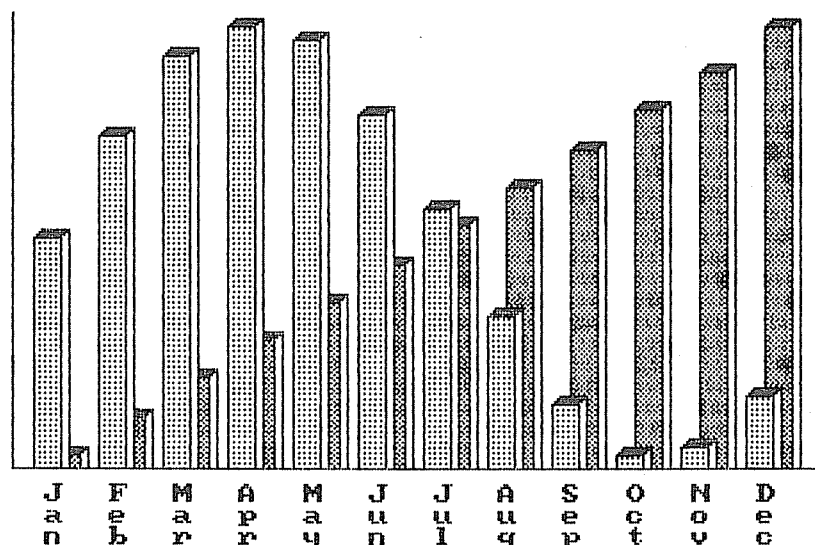


Fig. 12.14

Les différentes manières d'arranger un ensemble de valeurs dans un histogramme sont sans limites : les blocs peuvent être horizontaux, combinés avec des lignes, etc. Nous allons présenter ici trois exemples de diagrammes produits par de petites modifications du programme 12.9.

Le premier exemple utilise des lignes pour tracer deux ensembles de données. Pour différencier nettement les deux ensembles, une ligne trace les points limites avec des carrés, et l'autre avec des cercles. La figure 12.15 a été réalisée en fusionnant le programme 12.12 avec le programme 12.9.

```

300 CIRCLE(X+3,YT),2,3,,1:-(X+4,YT+1),3,B
310 IF I>1 THEN LINE(PXT+1,PYT)-(X+3,YT-1)
320 LINE(X-1,YS-1)-(X+1,YS+1),3,B
330 IF I>1 THEN LINE(PXS+1,PYS)-(X,YS-1)
335 PXT=X+3:PYT=YT:PXS=X:PYS=YS

```

PROGRAMME 12.12

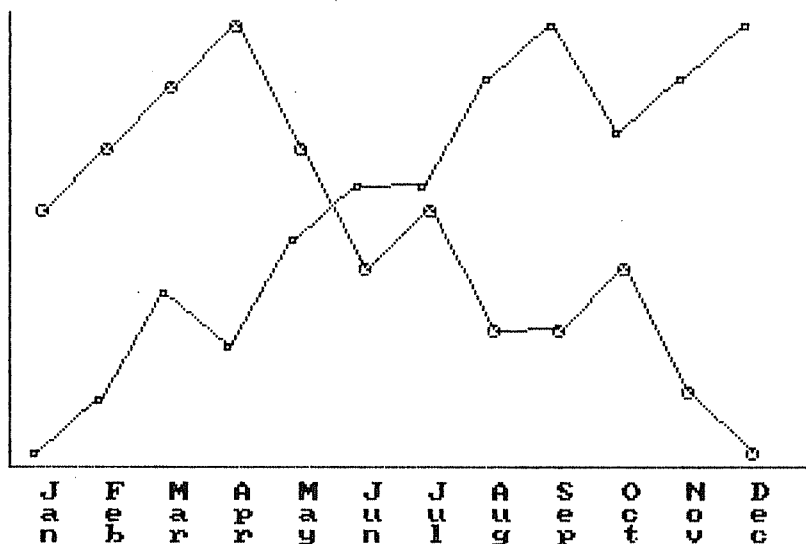


Fig. 12.15

Dans la figure 12.16, produite en fusionnant le programme 12.13 avec le programme 12.9, un ensemble de données est tracé avec une ligne, l'autre avec des barres.

```

300 LINE(X,YS)-(X+13,170),1,BF
310 LINE(X,YS)-(X+13,170),3,B
320 CIRCLE(X+7,YT),2,3,,1
330 IF I>1 THEN LINE(PXT+2,PYT)-(X+5,YT-1)
335 PXT=X+7:PYT=YT

```

PROGRAMME 12.13

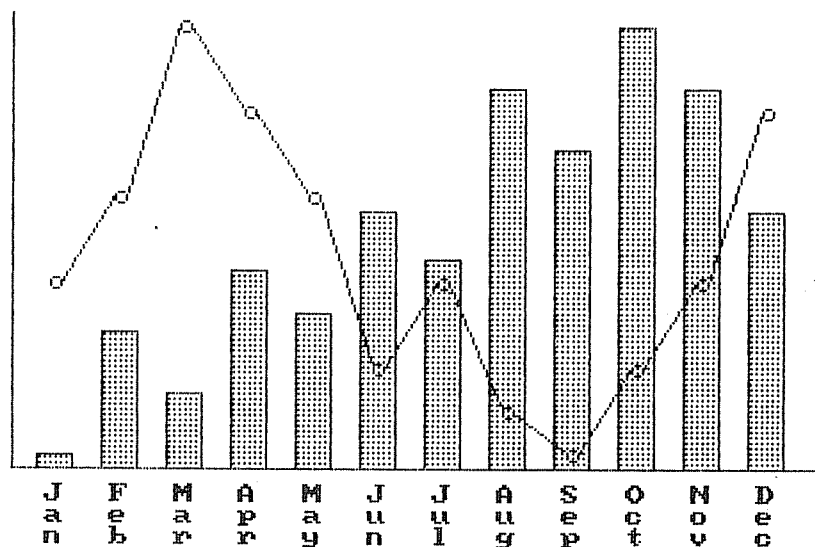


Fig. 12.16

Enfin, la figure 12.17 a été réalisée en effaçant les lignes 260 à 350 du programme 12.9 et en le fusionnant ensuite au programme 12.14. Un ensemble de données est représenté par une montagne colorée limitée en haut par une ligne qui varie suivant les valeurs, et en bas par l'axe des X, l'autre ensemble est tracé avec des barres.

```

260 FOR I=1 TO 12
270   X=5+I*24
280   YT=165-(T(I)-T(MINT))*FACT
290   YS=165-(S(I)-S(MINS))*FACS
330   IF I>1
331     THEN
332       LINE(PXT,PYT)-(X+5,YT)
332   IF I=1 OR I=12
333     THEN
334       LINE(X+5,YT)-(X+5,170)
335   PXT=X+5:PYT=YT
340 NEXT
350 PAINT(40,168),2,3
560 FOR I=1 TO 12
570   X=5+I*24
580   YT=165-(T(I)-T(MINT))*FACT
590   YS=165-(S(I)-S(MINS))*FACS
600   LINE(X,YS)-(X+13,170),1,BF
610   LINE(X,YS)-(X+13,170),3,B
640 NEXT
650 W$=INPUT$(1)

```

PROGRAMME 12.14

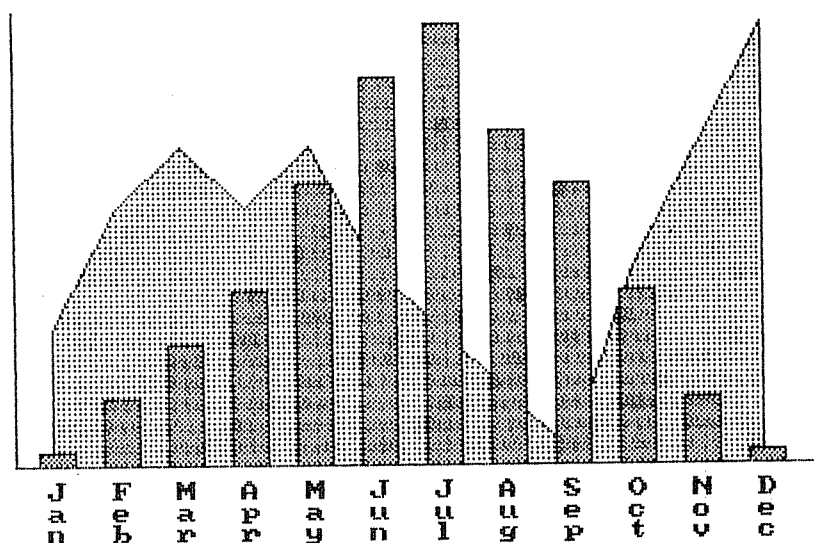


Fig. 12.17

12.7 LES CAMEMBERTS

Les camemberts sont une autre manière très prisée pour afficher des données. Aux paragraphes 3.2.5, 4.6 et 7.19 nous avons étudié la plupart des concepts dont nous avons besoin pour les générer. Il existe toutefois une méthode pour séparer les morceaux du camembert que nous n'avons pas encore mentionnée. Nous appelons cette méthode l'« éclatement » du camembert ! L'idée est d'appliquer une translation à des morceaux choisis de façon à ce qu'ils se déplacent du centre. A la différence des translations que nous avons étudiées dans les chapitres précédents, ici nous ne voulons pas déplacer toute la figure sur la gauche ou sur la droite, en haut ou en bas. Nous voulons décaler chaque morceau dans la direction de la ligne qui relie le centre du cercle au point qui divise l'arc de chaque morceau en deux. La figure 12.18 montre un camembert avec des flèches pointant dans les directions du mouvement pour chaque morceau.

Au programme 12.15, nous calculons les facteurs de décalage pour chaque morceau avec la même technique que nous avons utilisée pour trouver le point où démarrer le coloriage au paragraphe 4.6. Comme nous voulons un décalage modéré, nous avons choisi de prendre une fraction du rayon, et avec l'angle moyen (voir paragraphe 4.6) et les coordonnées polaires, nous trouvons les coordonnées des points se trouvant sur la ligne entre le centre et le milieu de l'arc de chaque morceau. Si ces valeurs sont additionnées à chaque X et Y lors de la création des mor-

ceux, l'effet sera similaire à une explosion dans laquelle chaque morceau est dirigé à l'extérieur d'une manière centrifuge, comme on peut le voir sur la figure 12.19. Notez que lorsqu'on calculait le Y de chaque facteur de décalage, on changeait son signe : ceci pour compenser l'inversion des coordonnées verticales de l'écran.

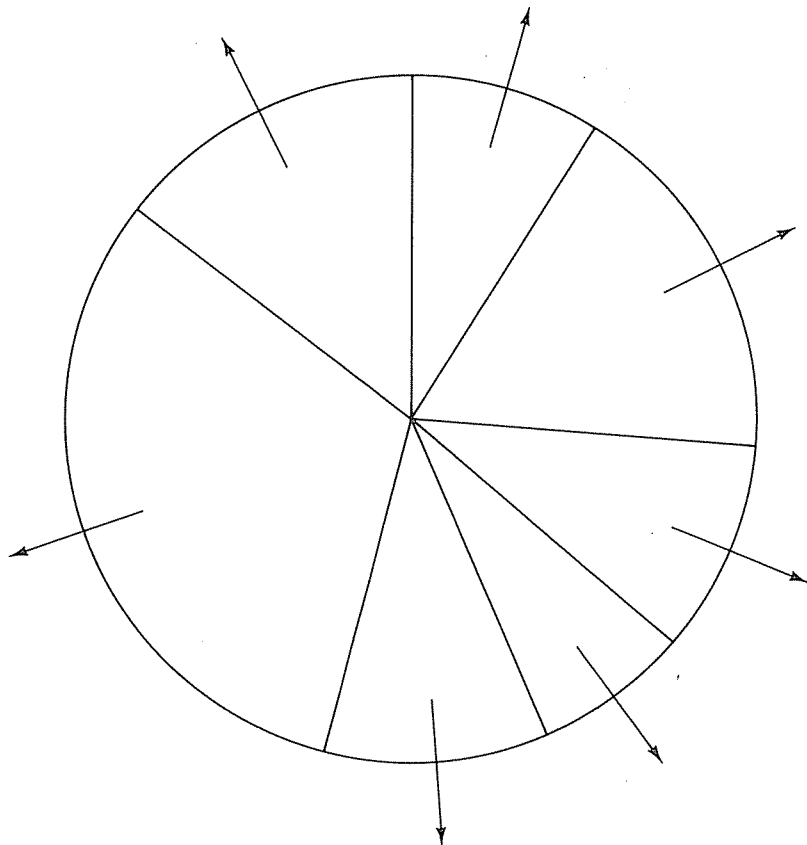


Fig. 12.18

```

0 '** 12-15 : Exploded pie **
10 SCREEN 1:CLS:DIM M(20),T$(20),X(20),Y(20)
20 DEF FN ANG(X)=2*PI*X/100
30 PI=3.141593:A=0:F=1.745329E-02
40 CENTERX=160:CENTERY=100
50 RADIUS=98:ASPECT=5/6
60 INPUT N
70 FOR I=1 TO N:PRINT"amount,title ";
80   INPUT M(I),T$(I)
90 NEXT
100 FOR I=1 TO N:
    T=T+M(I):
NEXT

```

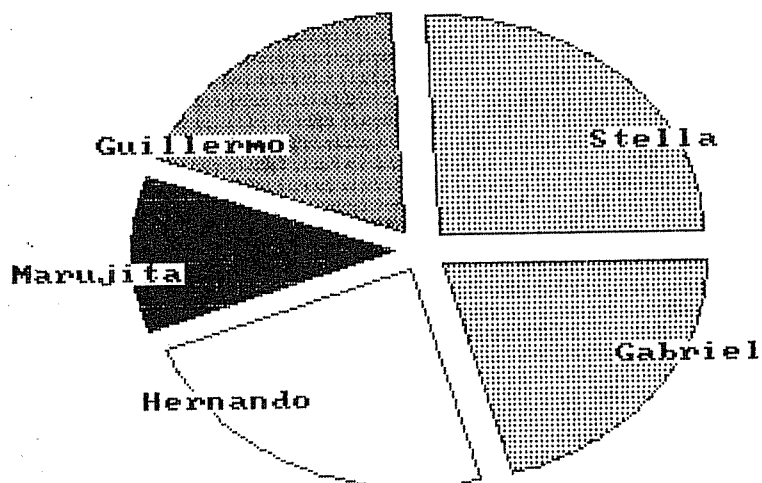


Fig. 12.19

```

110 CLS
120 FOR I=1 TO N:
    M(I)=M(I)*100/T:
NEXT
200 'Calculate offsets
210 A=0
220 FOR I=1 TO N
230   V=FN ANG(M(I))
240   IF A+V>6.283186
      THEN
        A=6.283186-V
250   ANG=(A+A+V)/2
260   X(I)=RADIUS*.1*COS(ANG)
270   Y(I)=-(RADIUS*.1*SIN(ANG)*ASPECT)
280   A=A+V
290 NEXT
400 'Draw and flood wedges
410 A=0
420 FOR I=1 TO N
430   V=FN ANG(M(I))
440   COL=I MOD 4
450   IF A+V>6.283186
      THEN
        A=6.283186-V
460   CIRCLE(CENTERX+X(I),CENTERY+Y(I)),
    RADIUS,COL,-A,-(A+V)
470   ANG=(A+A+V)/2
480   X=CENTERX+RADIUS*.5*COS(ANG)
490   Y=CENTERY+RADIUS*.5*SIN(ANG)*ASPECT
500   PAINT(X+X(I),200-(Y+Y(I))),COL,COL
510   CIRCLE(CENTERX+X(I),CENTERY+Y(I)),
    RADIUS,3,-A,-(A+V)
520   A=A+V
530 NEXT
600 'Place titles
610 A=0
620 FOR I=1 TO N

```

```

630 V=FN ANG(M(I))
640 IF A+V>6.283186
    THEN
        A=6.283186-V
650 ANG=(A+A+V)/2
660 X=CENTERX+RADIUS*.8*COS(ANG)
670 Y=CENTERX+RADIUS*.8*SIN(ANG)*ASPECT
680 LOCATE 1+(200-(Y+Y(I)))/B,1+(X+X(I))/B;
    IF (X+X(I))<CENTERX
        THEN
            PRINT STRING$(LEN(T$(I)),29);
690 PRINT T$(I);
700 A=A+V
710 NEXT
720 T=0
730 LINE(0,0)-(319,199),3,B
740 W$=INPUT$(1)

```

PROGRAMME 12.15

Nous allons maintenant appliquer cette technique à un seul morceau, pour mettre en valeur un article qui doit être distingué des autres. A la ligne 95 du programme 12.16 le numéro du morceau à éclater est rentré en tant que WB, et la boucle des lignes 220 à 290 calcule tous les facteurs de décalage. Quand les morceaux sont dessinés et coloriés (lignes 400 à 530), on ajoute XT et YT aux paramètres, à l'endroit où dans le programme 12.15 nous ajoutions X(I) et Y(I). A la ligne 425 on compare I avec WB, et le morceau est décalé. S'il n'est pas égal à WB, XT et YT sont mis à zéro et le morceau reste en place. On utilise le même processus pour placer les annotations. La figure 12.20 montre un exemple d'un morceau éclaté.

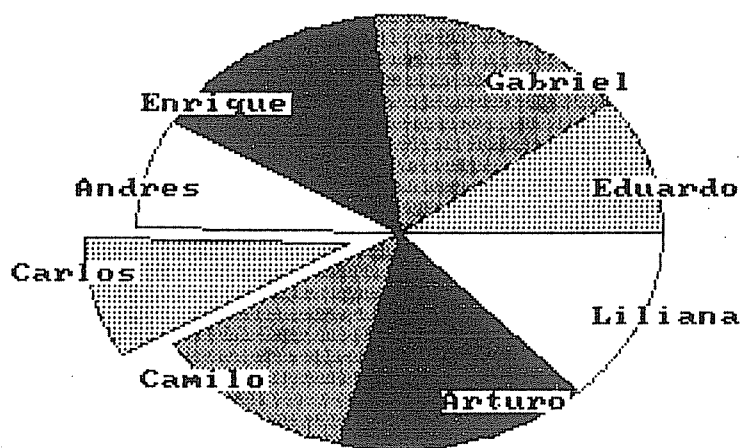


Fig. 12.20

```

0  ** 12-16 : Exploded wedge **
10 SCREEN 1:CLS:DIM M(20),T$(20),X(20),Y(20)
20 DEF FN ANG(X)=2*PI*X/100
30 PI=3.141593:A=0:F=1.745329E-02
40 CENTERX=160:CENTERY=100
50 RADIUS=98:ASPECT=5/6
60 INPUT N
70 FOR I=1 TO N:PRINT"amount,title ";
80   INPUT M(I),T$(I)
90 NEXT
95 INPUT"Wedge to blow ";WB
100 FOR I=1 TO N:
    T=T+M(I):
  NEXT
110 CLS
120 FOR I=1 TO N:
    M(I)=M(I)*100/T:
  NEXT
200 'Calculate offsets
210 A=0
220 FOR I=1 TO N
230   V=FN ANG(M(I))
240   IF A+V>6.283186
    THEN
      A=6.283186-V
250   ANG=(A+A+V)/2
260   X(I)=RADIUS*.2*COS(ANG)
270   Y(I)=-(RADIUS*.2*SIN(ANG)*ASPECT)
280   A=A+V
290 NEXT
400 'Draw and flood wedges
410 A=0
420 FOR I=1 TO N
425   IF I=WB
    THEN
      XT=X(I):YT=Y(I)
    ELSE
      XT=0:YT=0
430   V=FN ANG(M(I))
440   COL=I MOD 4
450   IF A+V>6.283186
    THEN
      A=6.283186-V
460   CIRCLE(CENTERX+XT,CENTERY+YT),
    RADIUS,COL,-A,-(A+V)
470   ANG=(A+A+V)/2
480   X=CENTERX+RADIUS*.5*COS(ANG)
490   Y=CENTERY+RADIUS*.5*SIN(ANG)*ASPECT
500   PAINT(X+XT,200-(Y+YT)),COL,COL
510   CIRCLE(CENTERX+XT,CENTERY+YT),
    RADIUS,3,-A,-(A+V)
520   A=A+V
530 NEXT
600 'Place titles
610 A=0
620 FOR I=1 TO N
625   IF I=WB
    THEN
      XT=X(I):YT=Y(I)
    ELSE
      XT=0:YT=0

```

```

630 V=FN ANG(M(I))
640 IF A+V>6.283186
    THEN
        A=6.283186-V
650 ANG=(A+A+V)/2
660 X=CENTERX+RADIUS*.8*COS(ANG)
670 Y=CENTERY+RADIUS*.8*SIN(ANG)*ASPECT
680 LOCATE 1+(200-(Y+YT))/8,1+(X+XT)/8:
    IF (X+XT)<CENTERX
    THEN
        PRINT STRING$(LEN(T$(I)),29);
690 PRINT T$(I);
700 A=A+V
710 NEXT
720 T=0
730 LINE(0,0)-(319,199),3,B
740 W$=INPUT$(1)

```

PROGRAMME 12.16

EXERCICES

1. Ecrire un programme dans lequel les barres sont tracées horizontalement.
2. Ecrire un programme dans lequel les données sont affichées simultanément dans un histogramme et dans un camembert en haute résolution.
3. Ecrire un programme dans lequel les annotations d'un camembert sont à l'extérieur du cercle, et connectées au morceau correspondant par une ligne.
4. Ecrire un programme dans lequel deux histogrammes sont montrés simultanément (comme dans la figure 12.12) ; sur la gauche l'échelle verticale devra indiquer en rouge les valeurs des barres rouges, et sur la droite une seconde échelle devra indiquer en vert les valeurs des barres vertes.

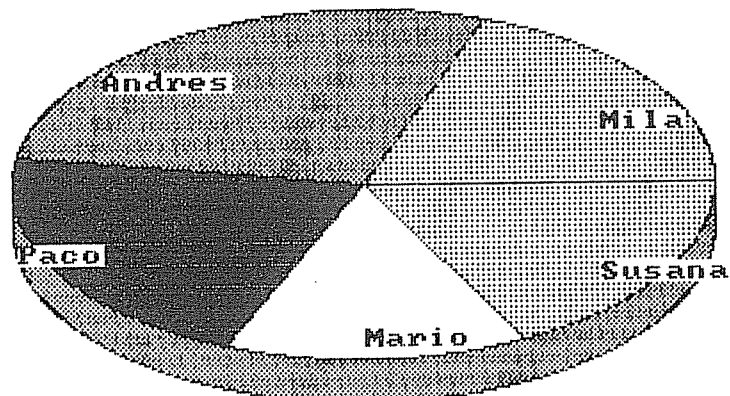


Fig. 12.21

5. On réalise un effet intéressant lorsqu'un camembert est incliné, comme le montre la figure 12.21.

Les arcs sont dessinés avec un petit *aspect*, et les calculs des points à partir desquels il faut commencer le coloriage et placer les annotations est modifié pour refléter ce changement. Pour dessiner le côté du camembert, l'arc de π à 0 (ou à 2π) est dessiné plusieurs fois, en commençant au centre en $CENTERX, CENTERY$, et en diminuant plusieurs fois (à $CENTERY + 1, +2, \dots$). Modifier le programme 12.16 de manière qu'un morceau du camembert soit éclaté.

NOTES

1. Si on utilise `WRITE` au lieu de `PRINT`, seuls les chiffres et le signe moins, si c'est le cas, seront imprimés.
2. Ceci a déjà été fait lorsqu'on traçait les fonctions au paragraphe 2.5.2.

APPENDICE A. LES NOMBRES BINAIRES

Le système numérique que nous utilisons pour les calculs de tous les jours est appelé système décimal, parce que chaque valeur est représentée en différents groupes de dix, et on utilise dix chiffres, de zéro à neuf. Voyons comment la position de chaque chiffre définit en clair sa valeur. Le nombre 703 signifie en fait

$$\begin{array}{r} 7 \times 100, \text{ ou } 7 \times 10^2 \\ + 0 \times 10, \text{ ou } 0 \times 10^1 \\ + 3 \times 1, \text{ ou } 3 \times 10^0 \end{array}$$

Le système binaire n'utilise que deux chiffres, le zéro et le un, pour représenter n'importe quel nombre. Son étude est importante parce que, à l'intérieur des ordinateurs, tout est représenté avec des zéros et des uns¹ : les nombres, les instructions et même les caractères et les chaînes. Dans l'état actuel de la technologie, il est plus facile de travailler avec seulement deux états différents (MARCHE ou ARRET, HAUT ou BAS, 0 ou 1) qu'avec 10.

Dans le système binaire, la position des chiffres indique différentes puissances de 2. Par exemple le nombre 703 sera représenté par 1010111111 en binaire. On peut voir la conversion en décimal à la figure A.1.

$$\begin{array}{r} 1 \times 512 \quad (512 = 2^9) \\ + 0 \times 256 \quad (256 = 2^8) \\ + 1 \times 128 \quad (128 = 2^7) \\ + 0 \times 64 \quad (64 = 2^6) \\ + 1 \times 32 \quad (32 = 2^5) \\ + 1 \times 16 \quad (16 = 2^4) \\ + 1 \times 8 \quad (8 = 2^3) \\ + 1 \times 4 \quad (4 = 2^2) \\ + 1 \times 2 \quad (2 = 2^1) \\ + 1 \times 1 \quad (1 = 2^0) \\ \hline \text{Total} = 703 \end{array}$$

Fig. A.1

Pour trouver l'équivalent binaire d'un nombre décimal, il faut faire plusieurs divisions entières (c'est-à-dire des divisions dont on écarte la partie fractionnaire) par les puissances de 2. La figure A.2 montre le processus pour trouver l'équivalent binaire du nombre décimal 61. L'opérateur « \ » comme dans A \ B équivaut à INT(A/B).

Dans les ordinateurs numériques, l'unité de base d'information est le bit, qui peut prendre une des deux valeurs : un ou zéro. Pour faciliter les choses, les bits sont groupés en octets, chacun comprenant huit bits.

61 \ 128 = 0	61 - 0 = 61
61 \ 64 = 0	61 - 0 = 61
61 \ 32 = 1	61 - 32 = 29
29 \ 16 = 1	29 - 16 = 13
13 \ 8 = 1	13 - 8 = 5
5 \ 4 = 1	5 - 4 = 1
1 \ 2 = 0	1 - 0 = 1
1 \ 1 = 1	1 - 1 = 0
61 in binary = 00111101	

Fig. A.2

Quand un octet a tous ses bits à 0 (00000000), sa valeur (décimale) est 0, quand tous sont des uns, sa valeur est 255.

Donc, un octet peut être utilisé pour mémoriser n'importe quelle valeur entre 0 et 255. Une liste partielle des équivalences entre les nombres binaires et décimaux est donnée à la figure A.3, où il est possible de se rendre compte de la forme particulière suivie par ces nombres.

En BASIC il existe six instructions pour travailler directement avec des octets :

ASC : Retourne la valeur (décimale) de l'argument.

CHR\$: Retourne le caractère qui correspond à l'argument.

PEEK : Retourne le contenu d'un certain octet de la mémoire.

POKE : Range la valeur d'un octet dans un emplacement de mémoire.

BSAVE : Transfère une partie de la mémoire sur disque (ou autre périphérique).

BLOAD : Transfère un groupe d'octets d'un disque (ou autre périphérique) dans la mémoire.

00000000 =	0	11110011 =	243
00000001 =	1	11110100 =	244
00000010 =	2	11110101 =	245
00000011 =	3	11110110 =	246
00000100 =	4	11110111 =	247
00000101 =	5	11111000 =	248
00000110 =	6	11111001 =	249
00000111 =	7	11111010 =	250
00001000 =	8	11111011 =	251
00001001 =	9	11111100 =	252
00001010 =	10	11111101 =	253
00001011 =	11	11111110 =	254
00001100 =	12	11111111 =	255

Fig. A.3

Représentation des nombres supérieurs à 255. Comme un octet ne peut mémoriser que des nombres allant jusqu'à 255, le moyen de mémoriser des nombres plus grands est normalement d'utiliser deux octets, dont on sait que l'un est multiplié par 255, permettant ainsi la représentation de nombres de 0 à 65535. La figure A.4 montre des exemples de ce type de représentation.

Premier octet	Second octet	Equivalence
1	0	$1 + 256^0 \cdot 0 = 1$
255	0	$255 + 256^0 \cdot 0 = 255$
0	1	$0 + 256^0 \cdot 1 = 256$
1	1	$1 + 256^0 \cdot 1 = 257$
254	255	$254 + 256^0 \cdot 255 = 65534$
255	255	$255 + 256^0 \cdot 255 = 65535$

Fig. A.4

Nombres hexadécimaux : Les nombres binaires ont une sorte d'effet hypnotique sur les humains. Une longue chaîne de zéros et de uns est extrêmement difficile à comprendre parce que nous perdons la piste de la position des chiffres. Un groupe de quatre nombres binaires (habituellement appelé un quartet) peut être représenté par un chiffre hexadécimal, ainsi un octet peut être représenté par deux de ces chiffres. Dans le système hexadécimal, chaque chiffre peut prendre une des seize valeurs possibles. Comme nous n'avons pas de symbole pour représenter les chiffres supérieurs à 9, on utilise les lettres de A à F. La figure A.5 montre les équivalents décimaux et binaires des seize chiffres hexadécimaux.

Pour convertir des nombres hexadécimaux en décimal, utilisez simplement ces valeurs, plus le fait que chaque chiffre doit être multiplié par la puissance de 16 appropriée. L'équivalent décimal du nombre hexadécimal 3C est

$$\begin{array}{r}
 3 \cdot 16^1 = 48 \\
 + C \cdot 16^0 = 12 \\
 \hline
 60
 \end{array}$$

Le nombre hexadécimal 4000, qui est la taille en octets de l'écran graphique, est en décimal :

$$\begin{array}{r}
 4 \cdot 16^3 = 16384 \\
 0 \cdot 16^2 = 0 \\
 0 \cdot 16^1 = 0 \\
 0 \cdot 16^0 = 0 \\
 \hline
 16384
 \end{array}$$

HEXADECIMAL	DECIMAL	BINAIRE
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Fig. A.5

NOTES

1. A l'intérieur de l'ordinateur il y a seulement des différences de tension hautes et basses, et non pas des uns et des zéros.

APPENDICE B. LES VARIABLES BOOLEENNES

A côté des trois variables numériques standards (entier, simple précision, double précision), il y en a une quatrième qui s'arrange pour rester cachée dans la plupart des programmes : le type booléen (ou logique). Sa particularité est qu'elle ne peut prendre que deux valeurs : VRAI ou FAUX. La représentation interne est -1 et 0 respectivement ¹. Comme n'importe lequel des trois types numériques peut mémoriser ces deux nombres, les valeurs booléennes peuvent être rangées dans n'importe quelle variable numérique.

Les valeurs de ce type sont en général le résultat de comparaisons ou d'expressions logiques. Vous êtes sûrement familiarisés avec ce type d'instructions

IF I>12 THEN..

où la partie THEN n'est exécutée que si la comparaison est VRAIE. C'est exactement là qu'une valeur booléenne est produite. La ligne suivante est moins évidente :

A = (I > 12):IF A THEN..

Les parenthèses autour I>12 ont été ajoutées pour séparer l'expression booléenne de l'instruction d'affectation, et ne sont pas obligatoires. Par exemple si I = 20 quand la suite d'instructions ci-dessus est exécutée, la variable A prend la valeur VRAI (-1) et l'instruction IF branche sur la partie THEN.

Opérateurs logiques. Les valeurs booléennes ont leur propre jeu d'opérations. Dans des instructions de ce type

IF (VALUE<300) AND (MONEY>1000) THEN..

une opération logique est accomplie. Celle-ci précisément peut être interprétée comme : « Si la première valeur booléenne est VRAI et la seconde est également VRAI, THEN... ». Il existe une méthode facile pour définir les opérateurs booléens, on l'appelle une table de vérité, et elle comprend les résultats de toutes les combinaisons possibles des opérandes. La table B.1 est la table de vérité pour l'opérateur AND (ET).

A partir de la table nous pouvons voir que la seule façon pour une opération AND (ET) d'avoir un résultat VRAI est d'avoir ses deux opérandes VRAI. A la suite vous avez les tables de vérité pour les cinq autres opérateurs logiques (OR, NOT, XOR, IMP et EQU).

Table B.1.

AND (ET)

A	B	A AND B
FAUX	FAUX	FAUX
FAUX	VRAI	FAUX
VRAI	FAUX	FAUX
VRAI	VRAI	VRAI

OR (OU)

A	B	A OR B
FAUX	FAUX	FAUX
FAUX	VRAI	VRAI
VRAI	FAUX	VRAI
VRAI	VRAI	VRAI

NOT (NON)

A	NOT A
FAUX	VRAI
VRAI	FAUX

XOR (OU EXCLUSIF)

A	B	A XOR B
FAUX	FAUX	FAUX
FAUX	VRAI	VRAI
VRAI	FAUX	VRAI
VRAI	VRAI	FAUX

IMP (IMPLIQUE)

A	B	A IMP B
FAUX	FAUX	VRAI
FAUX	VRAI	VRAI
VRAI	FAUX	FAUX
VRAI	VRAI	VRAI

EQU (EGALE)

A	B	A EQU B
FAUX	FAUX	VRAI
FAUX	VRAI	FAUX
VRAI	FAUX	FAUX
VRAI	VRAI	VRAI

Opérations logiques sur des valeurs non booléennes. On peut utiliser les six opérateurs logiques avec des valeurs entières standards pour exécuter des opérations logiques au niveau du bit. Voyons un exemple :

A = 3 (en binaire : 00000011)
B = 2 (en binaire : 00000010)

A AND B = 2 (en binaire : 00000010)

Chaque bit de A est comparé avec le bit correspondant de B. Si tous les deux sont des 1, le bit résultant est 1 (c'est pourquoi le second bit du résultat en partant de la droite est un 1) ; sinon le résultat est 0. L'opération est un AND (ET) booléen sur chaque bit, dans laquelle 1 est considéré VRAI et 0 FAUX. C'est le type d'opérations qui a lieu quand les valeurs rangées dans un tableau sont écrites (PUT) sur l'écran. La valeur de la couleur de chaque pixel (une valeur de 0 à 3) est logiquement accouplée avec la valeur de la couleur dans le tableau et l'opération logique est exécutée. Si par exemple la valeur d'un pixel sur l'écran est 3 et la valeur dans le tableau 1, on exécute l'opération AND sur ces deux valeurs (le tableau est écrit (PUT) sur l'écran) et le pixel résultant aura la couleur 1. La table suivante montre que les octets ayant seulement un bit à 1 sont tous des puissances de deux

00000001 = 1 (2 ↑ 0)
00000010 = 2 (2 ↑ 1)
00000100 = 4 (2 ↑ 2)
00001000 = 8 (2 ↑ 3)
00010000 = 16 (2 ↑ 4)
00100000 = 32 (2 ↑ 5)
01000000 = 64 (2 ↑ 6)
10000000 = 128 (2 ↑ 7)

Il est possible de mettre à 1 tous les bits d'un octet sauf un, avec une simple opération :

N = 2 (N = 00000010)
N = NOT (N) (N = 11111101).

Une opération OR (OU) peut mettre à 1 un bit particulier sans toucher au reste de l'octet.

N = 0 (N = 00000000)
N = N OR 32 (00000000
 OR 00100000
 00100000 = 32);

```

N = 109      (N = 01101101)
N = N OR 16  (   01101101
               OR  00010000
               01111101 = 125)

```

Pour vérifier si un bit particulier est à 1, on exécute l'opération AND sur cet octet et un autre nombre dont tous les bits sont à zéro, sauf celui qu'il faut tester.

```

N = 109      (N = 01101101)
N = N AND 4  (   01101101
               00000100
               00000100 = 4)

```

Si le résultat est différent de 0, le bit est à 1 sinon il est à 0.
Voici trois fonctions utiles pour travailler avec les bits :

```

10 DEF FN SET(BYTE,BIT)=BYTE OR 2^BIT
20 DEF FN UNSET(BYTE,BIT)=BYTE AND NOT 2^BIT
30 DEF FN SCAN(BYTE,BIT)=( (BYTE AND 2 ^BIT)<>0)

```

La fonction SET met à 1 le bit spécifié d'une valeur d'octet particulière sans toucher le reste des bits. Par exemple FN SET(0,3) est égal à 8, ou 00001000. Notez que les bits sont comptés à partir de la droite, en partant du bit 0.

La fonction UNSET remet à 0 le bit spécifié d'un octet (on ne peut pas utiliser RESET comme nom de la fonction parce que c'est un mot réservé). FN UNSET(8,3) = 0 tandis que FN UNSET(8,2) = 8.

La fonction SCAN retourne VRAI si le bit spécifié est 1 dans la valeur de l'octet donné, et FAUX sinon. FN SCAN(8,3) = VRAI (VRAI est égal à -1) alors que FN SCAN(8,5) = FAUX.

Avec ces fonctions, il est facile de déterminer les tables de vérité des opérateurs logiques :

```

A EQU B  est équivalent à (A AND B) OR (NOT A AND NOT B)
A XOR B  est équivalent à NOT (A EQU B)
A IMP B  est équivalent à A OR NOT B

```

Voici un programme qui utilise la fonction SCAN pour convertir un nombre décimal (entre 0 et 255) en sa représentation binaire.

```

0  "" Decimal to binary ""
10 DEF FN SCAN(BYTE,BIT)=(BYTE AND 2 ^BIT)<>0
20 CLS:INPUT N:W$=""
30 FOR I=0 TO 7

```

```
40  IF FN SCAN(N,I)
      THEN
        W$="1"+W$
      ELSE
        W$="0"+W$
50  NEXT
60  PRINT W$
```

NOTES

1. BASIC interprète 0 comme FAUX et n'importe quel autre nombre comme VRAI. Toutefois le VRAI résultant de comparaisons est toujours -1.

APPENDICE C. VIDAGE DE L'ECRAN SUR L'IMPRIMANTE

Nous vous présentons ici les trois programmes avec lesquels tous les graphiques de ce livre furent transférés de l'écran sur une imprimante EPSON FX-80.

```
0 ' ** C-1 : Dump in black & white **
32000 WIDTH "LPT1:",255:
      LPRINT CHR$(27)"@";:DEFINT A-Z
32010 OPEN"R",1,"LPT1:":WIDTH#1,255
32020 DIM PT(8):
      FOR I=0 TO 7:
        PT(I)=2^(7-I):
      NEXT
32030 LPRINT CHR$(27);"3";CHR$(24);
32040 LPRINT SPACE$(15);CHR$(27);"*";
      CHR$(5);CHR$(64);CHR$(1);
32050 FOR J=0 TO 199 STEP 8
32060   FOR X=0 TO 319
32070     FOR Y=0 TO 7
32080       IF POINT(X,J+Y)=0
          THEN
            32090
          ELSE
            T=T+PT(Y)
32090     NEXT
32100     PRINT#1,CHR$(T);:T=0
32110     NEXT:LPRINT:
      LPRINT SPACE$(15);CHR$(27);"*";
        CHR$(5);CHR$(64);CHR$(1);
32120 NEXT
32130 W$=INKEY$:
      IF W$=""
        THEN
          SOUND 2000,1:GOTO 32130
```

PROGRAMME C.1

```
0 ' ** C-2 : Dump with two levels of grey **
32000 WIDTH "LPT1:",255:
      LPRINT CHR$(27)"@";:DEFINT A-Z
32010 OPEN"R",1,"LPT1:":WIDTH#1,255
32020 DIM PT(8):
      FOR I=0 TO 7:
        PT(I)=2^(7-I):
      NEXT
32030 LPRINT CHR$(27);"3";CHR$(24);
32040 LPRINT SPACE$(15);CHR$(27);"*";
      CHR$(5);CHR$(64);CHR$(1);
32050 FOR J=0 TO 199 STEP 8
32060   FOR X=0 TO 319
32070     FOR Y=0 TO 7
32075       PO=POINT(X,J+Y)
32080       IF PO=0
          THEN
            32090
```

```

32082      IF PO=3
           THEN
             T=T+PT(Y):GOTO 32090
32085      IF PO=2
           THEN
             IF (X MOD 2=0 AND Y MOD 2=0)
               OR (X MOD 2=1 AND Y MOD 2=1)
             THEN
               T=T+PT(Y):GOTO 32090
             ELSE
               32090
32087      IF (X MOD 2=1 AND Y MOD 2=1)
           THEN
             T=T+PT(Y)
32090      NEXT
32100      PRINT#1,CHR$(T);:T=0
32110      NEXT:LPRINT:
           LPRINT SPACE$(15);CHR$(27);"*";
           CHR$(5);CHR$(64);CHR$(1);
32120      NEXT
32130      W$=INKEY$:
           IF W$=""
           THEN
             SOUND 2000,1:GOTO 32130

```

PROGRAMME C.2

```

0  ** C-3 : High-resolution dump **
32000      WIDTH "LPT1:",255:
           LPRINT CHR$(27)"@";:DEFINT A-Z
32010      OPEN"R",1,"LPT1:":WIDTH#1,255
32020      DIM PT(8):
           FOR I=0 TO 7:
             PT(I)=2^(7-I):
           NEXT
32030      LPRINT CHR$(27);"3";CHR$(24);
32040      LPRINT CHR$(27);"*";CHR$(4);
           CHR$(128);CHR$(2);
32050      FOR J=0 TO 199 STEP 4
32060        FOR X=0 TO 639
32070          FOR Y=0 TO 3
32080            IF POINT(X,J+Y)=0
              THEN
                32090
              ELSE
                T=T+PT(2*Y)+PT(2*Y+1)
32090          NEXT
32100          PRINT#1,CHR$(T);:T=0
32110          NEXT:LPRINT:
           LPRINT CHR$(27);"*";CHR$(4);
           CHR$(128);CHR$(2);
32120      NEXT
32130      W$=INKEY$:
           IF W$=""
           THEN
             SOUND 2000,1:GOTO 32130

```

PROGRAMME C.3

APPENDICE D. LES TOUCHES DE FONCTIONS

Les dix touches de fonctions situées à la gauche du clavier de l'IBM PC peuvent être d'un grand recours pour la programmation. Elles peuvent générer n'importe quelle chaîne allant jusqu'à quinze caractères y compris ESC, le retour chariot, les guillemets et les caractères de contrôle. La table D.1 montre les chaînes prises par défaut.

Table D.1.

F1 LIST	F6 , "LPT:"
F2 RUN	F7 TRON
F3 LOAD"	F8 TROFF
F4 SAVE"	F9 KEY
F5 CONT	F10 SCREEN 0,0,0

Pour affecter une chaîne à une touche de fonction, utilisez l'instruction KEY *n*,*chaîne* où *n* peut prendre les valeurs de 1 à 10 et *chaîne* peut être n'importe quelle chaîne allant jusqu'à 15 caractères. Par exemple après l'instruction KEY 8, « CLS » + CHR\$(13), chaque fois que la touche de fonction 8 est pressée, l'instruction CLS est générée comme si elle avait été tapée au clavier. Comme elle est suivie par un retour de chariot (CHR\$(13)), l'instruction est exécutée immédiatement.

Voici une liste des différences entre les chaînes produites par les touches de fonctions et celles générées par le clavier :

Le bit de poids fort (le plus à gauche), des caractères affectés aux touches de fonctions est toujours 0 (tous les nombres sont traités MODULO 127), ainsi CHR\$(128) générera CHR\$(0) et CHR\$(255) générera CHR\$(127).

Quand on appuie sur la touche ESC (CHR\$(27)) lors d'une entrée (INPUT), au niveau de l'instruction, ou quand ESC est généré par une touche de fonction, la ligne en cours est effacée. Si c'est imprimé à partir de l'intérieur d'un programme, elle n'est pas effacée.

Avec les touches de fonctions, il est impossible de générer les codes à deux caractères du clavier numérique. Si par exemple on affecte CHR\$(0) + CHR\$(119) (le code généré par la touche Ctrl-HOME) à une touche de fonction, il n'effacera pas l'écran comme le fait Ctrl-HOME.

La chaîne générée par la touche de fonction est imprimée à la position du curseur en cours. Si la chaîne contient une instruction et qu'elle est imprimée dans une ligne où il y a d'autres caractères, ceux-ci ne sont pas effacés et il est vraisemblable qu'une erreur sera signalée. Par exemple

si la chaîne « LIST » + CHR\$(13) a été affectée à une touche de fonction et que le curseur se trouve sur le premier caractère d'une ligne contenant la chaîne PRINT I1, si on appuie alors sur la touche de fonction on aura la chaîne LIST I1, qui n'est pas une instruction autorisée et provoque une « erreur de syntaxe », (« Syntax error »).

Si on ajoute un ESC avant l'instruction, le problème de caractères « fantômes » est évité parce que la ligne est vidée avant que l'instruction ne soit exécutée.

Les instructions qui doivent être exécutées immédiatement devront être suivies par un retour de chariot (CHR\$(13)) à la différence de celles qui exigent des paramètres. Les instructions qui exigent des noms de fichiers (comme LOAD et MERGE) devront se terminer par une seule apostrophe (CHR\$(14)).

Le programme D.1 montre une manière pratique de régler les touches de fonctions pour la programmation.

```
0 '** D-1 : Setting the function keys **
10 KEY 1,CHR$(27)+"Cl:"+CHR$(13)+"List "
20 KEY 2,CHR$(27)+"Run"+CHR$(13)
30 KEY 3,CHR$(27)+"Load"+CHR$(13)
40 KEY 4,CHR$(27)+"Save"+CHR$(13)
50 KEY 5,CHR$(27)+"Run"+CHR$(13)
60 KEY 6,"B:"
70 KEY 7,CHR$(27)+"Files"+CHR$(13)
80 KEY 8,"*.*"+CHR$(13)
90 KEY 9,CHR$(27)+"Screen 0"+CHR$(13)
100 KEY 10,CHR$(27)+"Edit "
```

PROGRAMME D.1

Instructions sur une seule touche. La plupart des lettres combinées avec la touche ALT se comportent comme des touches de fonctions pré-définies, ceci afin de pouvoir générer quelques instructions BASIC fréquemment utilisées en tapant seulement une touche. A la différence des touches de fonctions, elles ne peuvent pas être changées. La table D.2 montre les instructions produites par ces touches.

Table D.2.

A	AUTO	N	NEXT
B	BSAVE	O	OPEN
C	COLOR	P	PRINT
D	DELETE	Q	Q
E	ELSE	R	RUN
F	FOR	S	SCREEN
G	GOTO	T	THEN
H	HEX\$	U	USING
I	INPUT	V	VAL
J	J	W	WIDTH
K	KEY	X	XOR
L	LOCATE	Y	Y
M	MOTOR	Z	Z

APPENDICE E. LES INTERRUPTIONS EN BASIC

Aux paragraphes 7.1 et 7.2 nous avons vu une manière d'exécuter des instructions simples (comme allumer ou éteindre un curseur) tout en attendant une entrée. BASIC fournit une façon pratique pour communiquer avec les programmes sans toujours avoir à interroger le clavier. D'une manière semblable à ON ERROR GOTO, l'instruction ON KEY GOSUB se branchera au sous-programme indiqué quand la touche nommée sera tapée. Cette forme de communication est appelée une interruption. Le format de l'instruction est ON KEY(*n*) GOSUB *ligne*. *N* est une valeur comprise entre 1 et 14. Les touches correspondantes à chacune de ces valeurs sont données à la table E.1.

Table E.1.

VALEUR	TOUCHE
1 to 10	Touches fonctions de 1 à 10
11	Le curseur vers le haut
12	Le curseur à gauche
13	Le curseur à droite
14	Le curseur vers le bas

Ligne est le numéro de ligne sur lequel le programme se branchera quand on appuiera sur la touche. Le programme E.1 utilise les touches déplaçant le curseur à droite et à gauche pour incrémenter ou décrémente le nombre qui est en train d'être imprimé sur l'écran.

```
0 "" E-1 : Example of interrupts ""
10 CLS:D=5
20 ON KEY(12) GOSUB 1000
30 ON KEY(13) GOSUB 2000
40 KEY(12) ON:KEY(13) ON
50 LOCATE 10,18:PRINT X
60 X=X+D:GOTO 50
1000 D=-5:RETURN
2000 D=5:RETURN
```

PROGRAMME E.1

Après que les lignes 10 à 40 aient été exécutées, le programme rentre ce qui pourrait apparaître comme une boucle sans fin, entre les lignes 50 et 60. Toutefois, quand on appuie sur l'une des deux touches du curseur à droite ou à gauche, les lignes 1000 ou 2000 sont activées : la touche du curseur à gauche rendra D égal à -5 et les valeurs imprimées

diminueront. La touche du curseur à droite réincrémentera la valeur. Après un ON KEY GOSUB, une instruction KEY(*n*)ON doit être exécutée pour permettre le saut d'interruption. Un KEY(*n*)OFF correspondant l'annule. Supposons que vous vouliez déplacer une figure sur l'écran en utilisant les interruptions des touches du curseur. Déplacer une forme demande en général d'effacer, de calculer et de redessiner. Si ON KEY GOSUB est toujours en action et que la procédure de mise à jour de l'écran n'est pas terminée, une nouvelle interruption (de la part d'un utilisateur impatient, ce qui dans les jeux par exemple veut dire tous les utilisateurs) démarrera un nouveau cycle différé, laissant le précédent inachevé. Pour éviter cela, à l'entrée du sous-programme spécifié dans ON KEY GOSUB, un KEY OFF automatique est exécuté, et le KEY ON correspondant est exécuté au retour.

Après une instruction ON KEY(*n*)GOSUB, si on annule l'interruption avec KEY OFF, le programme l'ignorera lorsqu'on tapera la touche. L'autre manière d'annuler l'interruption en faisant KEY (*n*) STOP fait que le programme se rappellera si on tape une touche et ainsi le KEY(*n*) ON suivant provoquera immédiatement un saut au sous-programme approprié si la touche de fonction correspondante a été tapée et pas encore traitée.

Il y a plusieurs instructions d'interruption, la table E.2 montre leurs formats et leurs actions.

Table E.2.

INSTRUCTION	ACTION DECLENCHANTE
ON COM(<i>n</i>) GOSUB ligne	Donnée présente dans l'adaptateur de communication numéro <i>n</i>
ON ERROR GOTO ligne	Erreur
ON KEY(<i>n</i>) GOSUB ligne	Touche <i>n</i> tapée
ON PEN GOSUB ligne	Crayon lumineux activé
ON STRIG(<i>n</i>) GOSUB ligne	Bouton de la manette (<i>n</i>) pressé

APPENDICE F. LES CODES ASCII

0 - Nul	32 - Blanc	64 - @	96 - '
1 - ☐	33 - !	65 - A	97 - a
2 - ☐	34 - "	66 - B	98 - b
3 - ☐	35 - #	67 - C	99 - c
4 - ☐	36 - \$	68 - D	100 - d
5 - ☐	37 - %	69 - E	101 - e
6 - ☐	38 - &	70 - F	102 - f
7 - Sonnerie	39 - '	71 - G	103 - g
8 - ☐	40 - (72 - H	104 - h
9 - Tabulation	41 -)	73 - I	105 - i
10 - Saut de ligne	42 - °	74 - J	106 - j
11 - Point de départ (en haut à gauche)	43 - +	75 - K	107 - k
12 - Saut de page	44 - ,	76 - L	108 - l
13 - Retour de chariot	45 - —	77 - M	109 - m
14 - ☐	46 - .	78 - N	110 - n
15 - ☐	47 - /	79 - O	111 - o
16 - ☐	48 - 0	80 - P	112 - p
17 - ☐	49 - 1	81 - Q	113 - q
18 - ☐	50 - 2	82 - R	114 - r
19 - ☐	51 - 3	83 - S	115 - s
20 - ☐	52 - 4	84 - T	116 - t
21 - ☐	53 - 5	85 - U	117 - u
22 - ☐	54 - 6	86 - V	118 - v
23 - ☐	55 - 7	87 - W	119 - w
24 - ☐	56 - 8	88 - X	120 - x
25 - ☐	57 - 9	89 - Y	121 - y
26 - ☐	58 - :	90 - Z	122 - z
27 - ☐	59 - ;	91 - [123 - {
28 - Curseur à droite	60 - =	92 - \	124 -
29 - Curseur à gauche	61 - =	93 -]	125 - }
30 - Curseur en haut	62 - >	94 - ^	126 - ~
31 - Curseur en bas	63 - ?	95 - —	127 -

Imprimé en France. — JOUVE, 18, rue Saint-Denis, 75001 PARIS
N° 17064. Dépôt légal : Avril 1987
N° d'Editeur : 4638

